AFRL-IF-RS-TR-2002-208
Final Technical Report
August 2002

TASKABLE REACTIVE AGENT COMMUNITIES

SRI International

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J380

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

AFRL-IF-RS-TR-2002-208 has been reviewed and is approved for publication

APPROVED: [illegible]

NANCY A. KOZIARZ
Project Engineer

FOR THE DIRECTOR: James A. Collins

JAMES A COLLINS
Acting Technical Advisor
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE Aug 02	3. REPORT TYPE AND DATES COVERED Final May 98 – Jan 02

4. TITLE AND SUBTITLE TASKABLE REACTIVE AGENT COMMUNITIES	5. FUNDING NUMBERS C - F30602-98-C-0160 PE - 63760E PR - AGEN TA - T0 WU - 07
6. AUTHOR(S) Karen L. Myers, David L. Martin and David N. Morley	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) SRI International 333 Ravenswood Ave Menlo Park, CA 94025	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington, VA 22203-1714 AFRL/IFTB 525 Brooks Rd Rome, NY 13441-4505	10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2002-208

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Nancy Koziarz, IFTB, 315-330-2828, koziarzn@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

The focus of Taskable Reactive Agent Communities (TRAC) project was to develop mixed-initiative technology to enable humans to supervise and manage teams of agents as they perform tasks in dynamic environments. TRAC technology would enable users to task agent communities in a high-level language that provides both descriptions of goals to be satisfied and boundaries on agent behavior. The TRAC approach can be viewed as providing a form of high-level process management technology that enables flexible human control of agent communities. In particular, the TRAC framework is intended to support applications where processes are distributed and automatable, but where human guidance can improve performance and increase flexibility. The technical work for the project focused on identifying useful categories of guidance for directing agents, designing languages for expressing guidance, and developing techniques for operationalizing guidance into constraints that influence agent behavior. Because users may provide incompatible or unsatisfiable guidance to agents, the work also encompassed techniques for detecting and resolving conflicting guidance. The TRAC implementation was used as the basis for a demonstration system called TIGER (TRAC Intelligence Gathering and Emergency Response), which focused on the problem of multiagent intelligence gathering in the wake of a simulated natural disaster. Within TIGER, a human supervisor can delegate tasks to agents while providing guidance to control their runtime behavior.

14. SUBJECT TERMS agent teams, conflict guidance, and mixed-initiative	15. NUMBER OF PAGES 56
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

1 Introduction

1.1 Background

Agent technology provides great promise for increasing levels of automation in complex distributed applications. One significant impediment to the acceptability of this technology, however, has been concern that increased automation necessarily translates into reduced flexibility and loss of human control.

The agents research community has, for the most part, focused on the mechanics of building autonomous agents and techniques for communication and coordination among agents. In contrast, little attention has been paid to supporting human interactions with agents. Most agent frameworks lie at the extremes of the interaction spectrum, either assuming full automation by the agents with no means for user involvement, or requiring human intervention at each step along the way (i.e., *teleoperation* mode). Recently, however, there has been increased interest in agent systems designed specifically to support interaction with humans (e.g., [2, 3, 6, 23]).

The focus of SRI International's Taskable Reactive Agent Communities (TRAC) project was to develop mixed-initiative technology to enable humans to supervise and manage teams of agents as they perform tasks in dynamic environments. TRAC technology would enable users to task agent communities in a high-level language that provides both descriptions of goals to be satisfied and boundaries on agent behavior. During execution, the human would manage agent activities in accord with a level of involvement that best suits his or her individual needs.

The TRAC approach can be viewed as providing a form of high-level process management technology that enables flexible human control of agent communities. In particular, the TRAC framework is intended to support applications where processes are distributed and automatable, but where human guidance can improve performance and increase flexibility. In doing so, TRAC should yield more effective and adaptive problem solving than is supported by current agent systems, while producing solutions that better match the preferences of users.

1.2 Project Evolution

Our original formulation of the project encompassed two complementary lines of research. One involved the design of a policy-based approach for formulating guidance to direct the operations of agents. The second involved the design of new delegation techniques and organizational strategies for agent communities that would enable more effective use of agent resources. Our work in this second area was expected to leverage the Open Agent Architecture (OAA), a facilitator-based agent framework that supports rapid development of agent systems.

In support of the overall objectives of the Control of Agent-Based Systems (CoABS) program, we spent most of our resources within the first two years of the project engaged in three Technology Integration Experiments (TIEs). The OAA system was used extensively within these TIEs to provide critical integration capabilities for a range of CoABS technologies, thus providing a valuable service to the program. In addition, we contributed numerous agents for use within the TIEs that provided capabilities such as speech and language understanding, multimodal mapping, video display, and information retrieval. This work, however, limited the technical advances that we were able to make relative to our original project goals.

After completion of the three TIEs, we were directed by program management to focus on policy-based agent directability as our primary research objective. For this final technical report, we emphasize our research results in this area, which were developed primarily during the last 20 months of the project. Section 5 provides brief descriptions of the TIE work from the first two years of the project (along with our work on a fourth TIE during the second half of the project).

1.3 Agent Directability: Technical Approach

Our technical approach for agent directability is grounded in the notion of *policies* that express general and task-specific recommendations for agents as they execute assigned tasks. The project emphasized three types of policy: (a) strategy preferences that describe approaches to be used in executing assigned tasks, (b) adjustability of agent autonomy, and (c) customizable reporting of agent progress.

The technical work for the project focused on identifying useful categories of guidance for directing agents, designing languages for expressing guidance, and developing techniques for operationalizing guidance into constraints that influence agent behavior. Because users may provide incompatible or unsatisfiable guidance to agents, the work also encompassed techniques for detecting and resolving conflicting guidance. Issues related to user interface design, while important, were not a major thrust.

The project produced a prototype TRAC system built on SRI's Procedural Reasoning System (PRS) [8]. PRS provides a core agent capability grounded in a Belief-Desire-Intention model of agency [22]. The PRS technology is general-purpose and powerful, having been used in applications that include real-time fault diagnosis for the space shuttle, mobile robot control, and management of communications networks.

The TRAC implementation was used as the basis for a demonstration system called TIGER (TRAC Intelligence Gathering and Emergency Response), which focused on the problem of multiagent intelligence gathering in the wake of a simulated natural disaster. Within TIGER, a human supervisor can delegate tasks to agents while providing guidance to control their runtime behavior. TIGER served as a testbed for exploring our ideas on agent directability, as well as providing a key component of the CoABS MIATA TIE (Section 5).

1.4 Summary of Accomplishments

The main technical accomplishments for the project are listed below, along with references to those sections of the report that provide further detail.

- The definition of a policy framework for agent directability, focused on the complementary notions of strategy preference, adjustable autonomy, and customizable reporting (Section 2).

- The development of a framework for detecting and resolving conflicts related to user guidance for agents (Section 4).
- The implementation and documentation of a prototype TRAC system that supports our policy and conflict resolution methods [12], along with the development of the TIGER system that showcases TRAC within a multiagent intelligence-gathering domain (Section 5.2.2).
- Participation in four Technology Integration Experiments (TIEs) (Section 5).
- Several papers and reports describing our technical accomplishments on the project [12, 15, 16, 17, 18, 19] (Appendix A).

1.5 Report Overview

The main body of this report contains a high-level summary of the key technical contributions of the TRAC project. The appendix includes a selection of publications that provide more detailed technical information.

Section 2 describes our model of agent guidance. Section 3 describes two implemented tools designed to help users specify and apply agent guidance. Section 4 describes our work on detecting and resolving conflicts with agent guidance. Section 5 summarizes our involvement with CoABS TIEs. Section 6 presents our final conclusions and describes areas for future work.

2 Agent Directability

This section provides a high-level overview of our approach to agent directability. Additional technical details can be found in [16, 18].

We begin with a description of our underlying model of *agency* (Section 2.1), followed by a summary of our different categories of agent guidance (Section 2.2), an overview of the formal language used to express agent guidance (Section 2.3), and the semantics for the guidance language along with corresponding enforcement techniques (Section 2.4).

2.1 Agent Operations Model

We adopt a typical Belief-Desire-Intention (BDI) model of agency in the style of [22] (as embodied within SRI's Procedural Reasoning System (PRS) [8]). BDI agents are so-called due to the three components of their "mental state": *beliefs* that the agent has about the state of the world, *desires* to be achieved, and *intentions* corresponding to plans of action that the agent has adopted to achieve its desires.

A set of *plans* defines the range of activities that an agent can perform to respond to events or to achieve assigned tasks; our plan model is based on the Act representation [25]. Plans are parameterized templates of activities that may require variable instantiations to apply to a particular situation. The *cue* of a plan defines the reason for invoking a plan. A plan may be activated to respond to either a new goal (originating with the agent itself, another agent, or a human supervisor) or to a change in the agent's beliefs about the world. *Preconditions* associated with plans define gating constraints that must be satisfied in order for a plan to be applied. A plan is said to be *applicable* to an event (e.g., new goal or belief change) when the plan cue matches the stimulus, and the plan preconditions are satisfied in the current world state. Plans can be decomposed into other plans (thus providing a hierarchical model of activity), or can specify actions that agents can execute directly.

An agent's plan library will generally contain a range of plans describing alternative responses to new events. Sets of these plans may be *operationally equivalent* (i.e., they share the same cue and preconditions) but differ in the approach that they embody. To select among these alternatives, an agent may have some form of meta-control policy, including user guidance.

A BDI executor runs a continuous *sense-decide-act* loop to respond to changes in its operating environment. At the start of each cycle, the executor collects all new goals and changed beliefs. Next, it determines whether there are any plans in its library that are applicable to these events. From this set, it selects some subset for execution and creates intentions for them. Finally, the executor performs some bounded number of steps for each current intention.

Within this framework, agents make three main classes of decision:

D1 *whether to respond to new goals and events*

D2 *how to select among multiple applicable plans*

D3 *how to select instantiations for plan variables*

2.2 Agent Guidance

Our directability framework assumes that agents are capable of fully autonomous operation. More concretely, an agent's plan library covers the range of activities required to perform its assigned tasks. This assumption means that agents do not depend on the human supervisor to provide knowledge for task execution. Within this setting, guidance provides customization of agent behavior to suit the preferences of the human supervisor. In many applications, such guidance will enable superior performance, given that few plan libraries will reflect the full experience, breadth of knowledge, and reasoning capabilities that a human supervisor can bring to the decision-making process.

Our model of agent directability focuses on general and task-specific policies to influence the activities undertaken by agents in their execution of assigned tasks. In particular, we emphasize the areas of (a) adjustable levels of agent autonomy, and (b) strategy preferences that describe approaches to be used by an individual agent in executing assigned tasks.

Successful delegation further requires human visibility into agent activities. For this reason, our framework includes methods for *customizable reporting*, which allow agents to adjust the frequency and detail of information that they transmit regarding their status and progress on assigned tasks.

2.2.1 Adjustable Autonomy

We define the autonomy of an agent to be the extent to which it is allowed to make decisions (specifically, D1 – D3 from Section 2.1) on its own. In situations where activities are routine and decisions straightforward, a human may be content to delegate all problem-solving responsibility to an agent. However, in situations where missteps could have severe consequences, the degree of autonomy of an individual agent should necessarily be controllable by a human.

We are interested in domains where agents will generally need to operate with high degrees of autonomy. For this reason, we assume a *permissive* environment: unless stated otherwise, agents are allowed to operate independent of human interaction. Our approach allows the human to adjust the scope of operations that can be undertaken by an agent on its own terms, focusing on the notions of *permission requirements* for action execution and *consultation requirements* for decision making.

Permission Requirements Permission requirements declare conditions under which an agent must elicit authorization from the human supervisor before executing actions. For example, the directive "Obtain permission before abandoning survey tasks with Priority > 3" imposes the constraint that an agent request approval from the supervisor to abandon a certain class of tasks.

Consultation Requirements Consultation requirements designate a class of agent decisions that should be deferred to the human supervisor. These decisions can relate to either the selection of a value for variable instantiation (e.g., “Consult when selecting locations for staging bases”) or the selection of a plan for a goal (e.g., “Consult when choosing a response to a failed survey task”).

Our model of permission and consultation requirements, like earlier work on authority models, provides the means to block performance of certain actions by an agent. However, authority models are generally static (e.g., the *levels of autonomy* in [2]) and often derived from organizational structures. In contrast, our approach provides a rich language for expressing permission and consultation policies, which can vary throughout a problem-solving session.

2.2.2 Strategy Preference

Strategy preferences express recommendations on how an agent should accomplish tasks. These preferences could indicate specific plans to employ or restrictions on plans that should not be employed, as well as constraints on how plan variables can be instantiated. Thus, for instance, political or resource allocation decisions made at a higher level of command may make certain strategies preferable; alternatively, a given commander may have favored approaches for accomplishing different types of task.

For example, a human directing a collection of intelligence-gathering agents might declare the following sorts of strategy preferences. The directive “Try contacting Nongovernmental Organizations for information before sending vehicles to towns on the west coast” expresses a preference for selecting among operationally equivalent plans. The directive “Only use helicopters for survey tasks in sectors that are expected to be inaccessible by truck for more than 1 week” restricts the choice of resource type for instantiating certain plan variables.

2.2.3 Customizable Reporting

The degree of visibility required by a human supervisor into the activity of a particular agent will necessarily vary according to circumstances. For example, during the initial stages of disaster relief effort, periodic detailed reports may be desired to provide a clear overall picture of operations. As crises arise, immediate reports on the status of certain activities may become essential; in addition, detailed status reports could be replaced by high-level summaries since the human supervisor already has a good understanding of the lay of the land.

Our model of *customizable reporting* enables a user to tailor the amount and type of information produced by agents to his or her evolving needs. We define different classes of reporting methods that can be activated by the human as appropriate. Dimensions of adjustability include the context in which events should be reported, frequencies for periodic reports, and level of detail.

2.3 Representation of Guidance

The guidance language builds on three main constructs: the BDI model of agency (Section 2.1), the underlying *agent domain theory* that defines the core capabilities of an agent, and a *domain metatheory* that highlights key semantic differences among elements in the domain theory. Details on our language for representing agent guidance can be found in [18]; here, we provide a high-level description.[1]

A standard domain theory for an agent consists of four types of basic element: *individuals* corresponding to real or abstract objects in the domain, *relations* that describe characteristics of the world, *goals* that an agent may adopt, and *plans* that describe available means for achieving goals.

The domain metatheory provides an abstracted characterization of elements of the domain theory that highlights key semantic differences. The role of the domain metatheory is to provide a high-level language for describing activity that abstracts from the details of an agent's internal representations. As discussed in [14], such a metatheory can provide a powerful basis for supporting user communication. The main concepts within our metatheory for agent guidance are *features* and *roles* (similar in spirit to those of [13]) defined for agent plans and goals.

A *plan feature* designates an attribute of interest for a plan that distinguishes it from other plans that could be applied to the same task. For example, among plans for route determination, there may be one that is `OPTIMAL` but `SLOW` with a second that is `HEURISTIC` but `FAST`; each of these attributes could be modeled as a feature. Although the two plans are operationally equivalent (i.e., same cue and preconditions), their intrinsic characteristics differ significantly. Features provide the means to distinguish among such alternatives.

A *plan role* describes a capacity in which a domain object is used within a plan; it maps to an individual variable within a plan. For instance, a route determination plan may contain variables `location.1` and `location.2`, with the former corresponding to the `START` and the latter the `DESTINATION`. Roles provide a semantic basis for describing the use of individuals within plans that abstracts from the details of specific variable names.

In analogous fashion, we can define features and roles for goals.

The value of the domain metatheory lies with its provision of a semantically motivated abstraction of the underlying planning domain. This abstraction builds on semantic linkage among domain elements, specifically the sharing of roles and features among multiple plans and goals. A domain metatheory would be developed in conjunction with the definition of the underlying domain theory for an agent. As discussed in [14], a domain metatheory should be a natural by-product of a principled approach to domain modeling.

Using a combination of the agent domain theory and the domain metatheory, a user can specify abstract classes of activities and goals, which are used as the basic building blocks for agent guidance. For example, an agent's involvement in survey tasks would be captured by indicating that the agent is executing a plan with the feature `Survey` (of which there could be many). The activity of abandoning a task with priority greater than 5 could be characterized as a plan with the feature `ABANDON` and role `CURRENT-TASK` such that the value bound to the variable corresponding to `CURRENT-TASK` satisfies the constraint `(> (TASK-PRIORITY CURRENT-TASK) 5)`.

Guidance is created by composing such abstract descriptions. Specifically, each piece of guidance consists of

[1]The guidance representation language supports encoding of domain-independent strategy preference and adjustable autonomy directives. Customizable reporting is realized within a separate set of domain-specific protocols, and is not discussed further in this report.

- An *agent context* that describes conditions on the Beliefs, Desires, and Intentions currently held by an agent
- A *recommendation* that describes how the agent should respond when in a situation that matches the specified context

Each type of guidance (e.g., strategy preference, permission requirement, consultation requirement) employs a different type of recommendation. For strategy preference guidance, the recommendation constitutes a description of activities to be either preferred or avoided when in the designated context. For permission requirement guidance, the recommendation describes activities for which the agent should seek approval from the human supervisor. For consultation requirements, the recommendation describes decisions that an agent should pass along to the supervisor.

2.4 Basic Model for Guidance Semantics and Enforcement

Satisfaction of an individual guidance rule is defined relative to the choice of plan for responding to a new goal or belief change event within a given BDI executor cycle. A guidance rule is deemed *relevant* to the plan selection process for a given event iff the *agent context* matches the current BDI state of the agent. In the event that a guidance rule is not relevant to a particular plan selection decision, then the guidance is trivially satisfied. Otherwise, satisfaction is characterized as follows. For a strategy preference rule, the selected plan must match the recommended activities. For a permission requirement rule, either the selected plan does not match the permission-constrained activity in the rule or the supervisor agrees to allow execution of the plan. For a consultation rule, either the designated decision did not surface within the plan selection process or the selected plan is consistent with the human supervisor's decisions.[2]

From the perspective of a BDI executor, guidance serves as a *filter* on the plan instances that an agent can execute. When a standard BDI agent attempts to find an instance of a plan from its library to apply to an event, it determines a set of applicable plan instances based on the plan cues and preconditions. The executor limits this set further by eliminating options that violate the above notions of guidance satisfaction for the current set of guidance.

Enforcement of guidance is attained through a simple modification to the standard BDI interpreter loop at the point where a plan instance is selected in response to a posted event. First, the agent's BDI state is matched to the agent context of all currently defined guidance to determine the relevant guidance for the current execution cycle. The relevant strategy preference rules are then used to eliminate plan instances that do not match their recommended activities. The resultant list is then traversed in order to find the first for which either the plan instance is not affected by relevant permission or consultation requirement rules, or queries to the human supervisor elicit any required execution permissions and decision values. The agent then applies the selected plan instance to the current goal.

This filter-based semantic model presents a simple and intuitive approach for interpreting guidance. However, the model has certain limitations. One interesting consequence of this model is

[2]The formal definitions for guidance satisfaction can be found in [18].

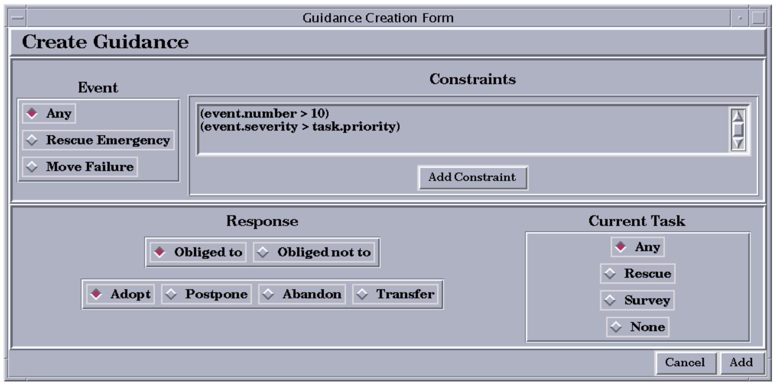
that a guidance rule that is not *relevant* to the current decision cycle is *trivially satisfied*. As discussed further in Section 4.1.2, this semantics can lead to some unintuitive results. In addition, the model does not provide an adequate solution to accommodate guidance that makes contradictory recommendations. In particular, contradictory recommendations would lead to no plan being selected for application to the current goal. Section 4 presents a more complex semantic model that provides a solution to these two problems.

3 Guidance Interface Tools

The motivation for our work on agent directability is to enable users to direct and manage agents in dynamic, unpredictable environments. The language presented in earlier sections provides a highly expressive formalism in which to define agent guidance; however, the complexity of the language could overwhelm a typical user. For this reason, we have developed tools to help users define and manipulate agent guidance. Figure 1 presents two such tools from the TIGER system.

The first tool is a *guidance authoring interface* that walks the user through the process of constructing a complex piece of guidance. To enable a simple specification process, the tool does not support the full expressivity of the formal guidance language; however, it supports a broad range of expressions, including the examples used within MIATA. An accompanying *guidance library* can be used to store authored guidance. Users can select guidance from the library, as appropriate for a particular situation.

The second tool is a *permissions window* that enables users to activate and deactivate permission requirements for certain classes of action performed on certain types of task. In particular, selections made through this interface are compiled into corresponding permission requirement structures. While this interface limits the scope of permission requirements that can be expressed, it provides a simple, accessible specification mechanism.


Guidance Creation Form

Create Guidance

Event: [✓ Any] Rescue Emergency / Move Failure

Constraints:
(event.number > 10)
(event.severity > task.priority)

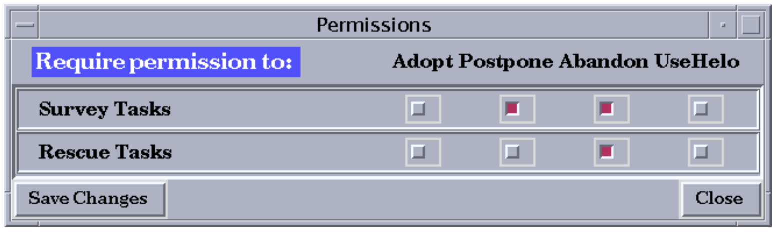
Add Constraint

Response: [✓ Obliged to] Obliged not to

[✓ Adopt] Postpone Abandon Transfer

Current Task: [✓ Any] Rescue Survey None

Cancel Add


Permissions

Require permission to:	Adopt	Postpone	Abandon	UseHelo
Survey Tasks	☐	☑	☑	☐
Rescue Tasks	☐	☐	☑	☐

Save Changes Close

Figure 1: Guidance Authoring Tool *(top)* and Permissions Window *(bottom)*

4 Guidance Conflicts

User guidance provides a powerful mechanism for runtime customization of agent behavior. However, it also introduces the potential for problems in the event that the guidance recommends inconsistent responses. Such conflicts cannot arise with adjustable autonomy guidance, but are a significant issue for strategy preference guidance. Robustness of operations requires mechanisms for detecting conflicts with strategy preference guidance and responding in a manner that does not jeopardize agent stability.

This project addressed two main issues related to the topic of conflicting strategy preference guidance for agents. First, we identified different types of conflict that can arise with strategy preference guidance. Second, we defined automated techniques for resolving guidance-related conflicts. Our approach combines the selective dropping of problematic pieces of guidance with a proactive capability to eliminate the source of conflicts by modifying current agent activities.

Our work did not consider interactive techniques for conflict resolution. However, such techniques would necessarily play an important role in a comprehensive conflict resolution system.

4.1 Types of Guidance Conflict

Guidance can lead to two types of conflict: *plan selection* and *situated guidance*.

4.1.1 Plan Selection Conflict

A plan selection conflict occurs when multiple pieces of guidance make incompatible recommendations for responding to a goal within a given cycle of the BDI executor. Conflicts of this type can arise in different forms. Here, we distinguish between *direct* and *indirect* conflicts.

A *direct conflict* arises when guidance yields contradictory plan selection recommendations. At the plan level, such conflicts can arise through explicitly contradictory directives (e.g., guidance that reduces to the constraints *Execute plan P* and *Don't execute plan P*), or implicitly due to in-place control policies (e.g., guidance that reduces to the constraints *Execute plan P* and *Execute plan Q* in the context of a control policy that allows only one response to any posted goal). Conflicts can also arise at the level of variable bindings (e.g., *Instantiate role R to A* and *Instantiate role R to B*, where $A \neq B$).

An *indirect conflict* among guidance occurs when there is no direct conflict, yet the plans recommended by the guidance cannot complete successfully because of interplan interactions. Such a situation could arise due to future contention for resources, deadlock/livelock, or race conditions (among others). The problem of indirect conflict arises for any multithreaded system, not just systems in which guidance has been used to select activities.

Direct conflicts are easily detected, as they lead to incompatible recommendations for responding to a belief change or posted goal. In contrast, it is generally difficult (sometimes impossible) to detect *a priori* the plan interference problems that underlie indirect conflicts. Because such interference problems remain an open research area in the agents community, we did not consider indirect conflicts within this project.

4.1.2 Situated Guidance Conflict

Our basic semantic model from Section 2.4 interprets guidance as a filter on the set of otherwise applicable plan instances for a particular goal or event. For example, consider a situation in which all TIGER vehicles are in use for various tasks, and that the human supervisor has asserted guidance q indicating that "All medical emergencies should be handled immediately". The relevant plans for responding to such emergencies each require the availability of a vehicle to perform various activities. Suppose an emergency event arises. Because all vehicles are in use, no immediate response plans would be applicable to the event. According to the filtering semantic model for guidance, q will have no effect in this situation. Had there been a vehicle available, however, an emergency response of some form would have been initiated.

In this case, there is a clear expectation on the part of the human supervisor for the system to respond to the emergency. Accommodation of this response requires a generalization of the filter-based semantic model described in Section 2.4.

More generally, this type of conflict arises in situations where there is a plan p that is relevant for a current goal g but some precondition C of p does not hold, making it inapplicable. The unsatisfied condition may be blocked by a contradictory belief of the agent, or as a result of some already executing activity. This type of situation can arise independent of the presence of guidance. Our interest in such situations relates to cases where guidance would recommend the execution of p but the violation of C results in the guidance being ignored because the plan(s) that it would recommend are not candidates for application.

From a more proactive perspective, the prior activity or state condition conflicts with the intent of applying the guidance. For this reason, we call this phenomenon a *situated guidance conflict,* as it depends on the consideration of guidance within a particular execution state of an agent. As with plan selection conflicts, situated conflicts can be further categorized according to whether they are direct or indirect.

4.2 Conflict Resolution Techniques

The basic semantic model for guidance interpretation (described in Section 2.4) rests on the notion of filtering plan instances during the BDI decision cycle that violate current guidance. This filter-based semantic model has the virtue of simplicity; however, it eliminates the applicability of guidance in many interesting situations.

One problematic situation arises when guidance makes contradictory recommendations, as described above for the case of plan selection conflicts (e.g., *"Execute plan P"* and *"Don't execute plan P"*). Because the filter-based semantic model eliminates plans that violate current guidance, such a situation would lead to the selection of no plan. As noted above, the filter-based semantics also leads to trivial satisfaction of guidance in cases where a more proactive interpretation would be preferred.

Satisfactory resolution of guidance conflicts requires a richer semantic model for guidance satisfaction. Our solution builds on the definition of satisfaction of an individual piece of guidance, as defined in [16, 18] (and described informally in Section 2.4 of this report). However, we adopt a preference-based approach that seeks to maximize satisfaction of a set of guidance relative to stated priorities. Furthermore, instead of reducing the set of plans that the agent considers (by filtering plans that would violate guidance), we expand the set of plans to include certain options that would otherwise be discarded as inapplicable in the current execution state.

The expansion is grounded in plans from the agent's predefined plan library. A dynamic synthesis process creates new plans from inapplicable plans that guidance might otherwise recommend by adding actions that compensate for the violated applicability conditions. To provide focus, the expansion process requires the satisfaction of prerequisites related to *resolvability* and *cost-benefit analysis*.

Resolvability The unsatisfied applicability conditions of the guidance-recommended plan must be *resolvable*. In particular, there must be identified methods that can be invoked to achieve the unsatisfied conditions. We refer to such methods as *resolution plans*. Resource availability constitutes one important class of potentially resolvable conditions; we focused on conditions related to serially sharable resources within the project.

Resolution plans are a form of goal-invoked plan designed to achieve a designated condition (e.g., the availability of a resource). Resolution plans may initiate actions to modify existing activities as part of the process of attaining the designated condition; in doing so, they must ensure that an agent is left in a stable state. Ensuring such stability can be a challenge in itself; we discuss that issue further in [20].

Cost-Benefit Analysis The benefits in following the prescribed guidance must outweigh the costs associated with execution of the resolution plans. The value in modifying existing activities to enable activation of new guidance-recommended activities depends on a range of factors. We developed a model grounded in the following criteria:

- *Resolvability Cost:* the cost inherent to executing any required resolution plans
- *Task Priorities:* the relative priorities of the resolution task and the current task

Different approaches can be considered for combining the above factors to determine the appropriate response to a situated guidance conflict. Within the TIGER system, we defined a set of fixed policies (e.g., *the priority of the new task is greater than that of the modified task*).

Details on our conflict resolution strategies can be found in [20].

5 Technology Integration Experiments

Our project played significant roles in three Technology Integration Experiments (TIEs) during the CoABS program:

- *People Finder/Mover* (NEO TIE 2)
- *Air Mobility* (NEO TIE 3)
- *Mixed-Initiative Agent Team Administration* (MIATA)

We also played a smaller role within the *Helicopter Evacuation* TIE (NEO TIE 1). Section 5.1 describes our involvement in the NEO TIEs, while Section 5.2 describes our work on MIATA.

5.1 NEO TIEs

From the beginning of the project through the end of 1999, the largest part of our work was in connection with our central role in defining and implementing the NEO (Non-combatant Evacuation Operation) TIEs (Technology Integration Experiments).

In September 1998, we attended the Challenge Problem Working Group meeting in Los Angeles, at which participants were asked to provide a list of individual technologies or “boxes” they could contribute to TIE demonstrations in the NEO domain. The result of the meeting was that participants organized themselves into three TIEs. As it turned out, SRI’s Open Agent Architecture (OAA) was selected as a principal building block for two of the TIEs. TIE 2 chose OAA as the primary agent infrastructure in which to bring together the various technology components, and TIE 3 developed as a collaborative effort to show interoperability of OAA with CMU’s RETSINA agent architecture [24].

In addition to providing and supporting the use of OAA as infrastructure for these TIEs, SRI made major contributions to the development of scenarios and system design, and contributed and evolved a number of individual components for both TIEs. In TIE 2, we collaborated closely with ISI, CMU/Veloso, Object Consulting, and others; in TIE 3, with CMU/Sycara and BBN.

The following subsection provides a brief overview of OAA. Next, we describe the use of OAA as infrastructure for both TIEs, and our work in supporting its use by others. Following that, we discuss the components and other contributions made to TIEs 2 and 3 individually, and summarize the major points demonstrated in these TIEs. In the final subsection of this section, we describe our efforts in moving these two TIEs onto the CoABS Grid.

5.1.1 Overview of OAA

OAA is a framework for constructing agent-based systems that makes it possible for software services to be provided through the cooperative efforts of distributed collections of autonomous agents [10, 4]. Communication and cooperation between agents are brokered by one or more *facilitators*, which are responsible for matching requests, from users and agents, with descriptions of the capabilities of other agents. Thus, it is not generally required that a requester (user or agent) know the identities, locations, or number of other agents involved in satisfying a request. Facilitators are not viewed as centralized controllers, however, but rather as *coordinators*, as they draw upon knowledge from several different, potentially distributed, sources to guide their delegation choices.

OAA is structured so as to minimize the effort involved in creating new agents and 'wrapping' legacy applications, written in various languages and operating on various platforms; to encourage the reuse of existing agents; and to allow for dynamism and flexibility in the makeup of agent communities. Distinguishing features of OAA as compared with related work include extreme flexibility in using facilitator-based delegation of complex goals, triggers, and data management requests; agent-based provision of multimodal user interfaces; and built-in support for including the user as a privileged member of the agent community.

OAA has to date been used as infrastructure for several dozen implemented systems, with widely varying requirements and domains, combining more than 100 agents, written in seven programming languages, on four different hardware platforms, as part of research and commercial efforts within and outside of SRI.

5.1.2 Infrastructure and Support for TIEs

Because OAA was used in two of the three NEO TIEs, a large amount of effort went toward supporting the participants who constructed components for these demonstrations. In addition to providing the OAA Facilitator component, we supplied OAA libraries to participants for a number of languages and platforms, and helped them with installation, development questions, and so forth. Our Java, C/C++, and Lisp libraries got the most significant use in these TIEs, and were used on both Unix and Windows platforms. This support began in late 1998 and continued throughout 1999.

Due to the strong interest by CoABS participants, we prepared a Web site with extensive documentation, a downloadable distribution of tools and libraries, a sample application, tutorial, and community exchange pages (e.g., mailing lists, FAQs) for OAA version 1.0. The distribution included new tools/agents for monitoring, debugging, profiling, and launching an agent community, which are discussed below. Although this Web site and distribution were initially password protected, it subsequently has been made public and has attracted interest from a wide variety of researchers in both government funded and commercially funded settings.

In addition to the ongoing TIE support activities described above, we made a substantial effort to provide a new port of our OAA Lisp library, running in Allegro Common Lisp on the NT platform. Also, in response to several suggestions and requests from TIE participants, we made selected upgrades to the functionality of the OAA Facilitator and libraries. In connection with the TIEs, the OAA Facilitator and libraries were successfully used by teams at Global Infotek, ISI, CMU (two different teams), BBN, and Object Consulting (and were also employed in lesser roles, in connection with TIE 1, at the University of Massachusetts and OGI). The feedback received from these teams was very positive and very useful.

Tools Although OAA was not developed under CoABS funding, two important agent development tools were: the Monitor agent and the Java version of Start-It. These tools were conceived to support our TRAC framework, but also turned out to be useful to a number of the NEO TIE participants, as well as to the developers of many OAA systems outside the scope of CoABS.

A key enabler of agent system development is the ability to inspect the state of individual agents and to monitor the communication flow among the community participants. To this end, the Monitor agent visualizes, logs, and constructs profiling information about multiagent interactions.

The Monitor agent turned out to be especially valuable in the context of remote testing (involving agents running at two or more sites) of interoperability between selected groups of TIE agents. Because the Monitor agent can remotely join an agent community at any time, and can display information about all agents in the community, and interactions between them, it enabled us to provide quick diagnoses of problems that arose in agents developed and running elsewhere. The Monitor agent also served as a example in designing the Grid visualization capabilities.

Start-It is a runtime tool that manages the execution state of an agent community. Start-It enables developers and other software agents to start, restart (if an agent unexpectedly terminates), and shut down agents in the community. The Start-It agent provides base-level infrastructure for experimenting with load balancing, resource discovery, and other approaches important for a constructing a dynamic, adaptable community of software agents.

5.1.3 TIE 2

The primary focus of TIE 2 was on the ability of an agent system to dynamically respond to changing information at runtime. The scenario and agent interactions were designed to explore the kinds of agent and infrastructure characteristics needed to

- Acquire new information sources at runtime (e.g., from the Internet)
- Monitor changes in information sources
- Rapidly interact with humans, giving them the right amount of information to help them be effective, in particular by displaying status changes and alerting humans to these changes
- Suggest appropriate responses
- Take commands from humans regarding actions to be carried out

In this TIE, we illustrated the following points:

- An agent-based paradigm provides a natural conceptual framework for designing and implementing complex, distributed software systems, and can improve productivity in developing such systems.
- Agents can monitor broad classes of information sources and dynamically identify relevant events.

- Humans can be kept ‘in the loop’ by tailoring the interface’s abstraction level to avoid information overload.
- Run-time coupling of heterogeneous components is feasible, and can increase the speed of assembling a complex system and its quality.
- Use of an agent-based paradigm can support greater reuse of legacy systems, with less effort than required by other paradigms.
- The use of facilitators in building complex, distributed software systems provides a useful approach, for many classes of system, to coordinating the efforts of relatively autonomous software components, which may not have been designed to work together.
- A multimodal, multimedia user interface provides a natural and easily acquired style of interaction with complex, distributed software systems, without requiring a user to be aware of the identities and activities of the individual software agents in the system.

TIE 2 Components For TIE 2, in addition to providing and supporting OAA as the framework of interoperation, we provided the Multimodal Map system [11], a telephone contact agent, and speech recognition and natural language understanding agents. Taken together, these agents, which work closely together through OAA mechanisms, provide a compelling means by which the operator can interact with the software components of the system. They allow the operator to give commands and ask queries using speech alone, or in conjunction with gestures on a map display.

While each of these components is written in a domain-independent, reusable fashion, some effort was required to customize them for the TIE (except for the phone agent, which required no modifications). For instance, the speech and natural language agents’ grammars had to be extended for the types of utterance used in the NEO domain. The most substantial new capabilities were those given to the Multimodal Map system, which included facilities for displaying and updating arbitrary routes on a map, and for analyzing the results of a database query to determine what information is appropriate for the map display, and to aggregate the data in such a way as to avoid an overly cluttered display.

Throughout the evolution of TIE 2, we made substantial contributions to scenario development, to the specification of interactions between the TIE agents, and to the development of the characterization of TIE 2 in terms of empirical hypotheses and experiments.

5.1.4 TIE 3

The primary focus of TIE 3 was an exploration of the challenges associated with establishing interoperability between two different agent frameworks – OAA and RETSINA – each of which, in turn, provides interoperability between multiple agents designed (or wrapped) to work within that framework. This work was documented in the paper [21].

To achieve the objectives of this TIE, we collaboratively designed and implemented an OAA-RETSINA “bridge” agent that enables arbitrary agents running in either framework to interoperate with agents running in the other framework, with minimal development effort required to introduce new agents. We also designed a scenario to illustrate the dynamic, adaptable aspects of agent architectures. Most of the implementation work was done at CMU.

Some of the key observations resulting from this exploration were as follows:

1. Building an interframework bridge agent can be useful, for some pairs of agent frameworks:
   - We found sufficient similarities between two independently developed frameworks to allow for a reasonably straightforward mapping between the frameworks' capabilities.
   - The interoperator isolates, localizes, and modularizes the critical functionality of translating messages between frameworks.
2. Building an interframework bridge agent is nontrivial:
   - Message translation involves semantic, not just syntactic, considerations.
   - There is no simple way to build an interoperator that can operate without some prior knowledge of the contents of each message type.
   - Further investigation is needed to determine what is required to build an interoperator without this prior knowledge.
   - There is likely to be some loss of functionality between frameworks (e.g., matchmaking based on argument types vs. based on unification).
3. Distributed development and testing of agent-based systems is feasible and productive, but a more mature approach to ontological issues is a critical need:
   - To minimize time spent in hand crafting solutions to ontological mismatches
   - To minimize task-specific prior knowledge required in the interoperator
4. A few basic tools make a big difference in agent system development efforts, for example:
   - Tracing mechanisms in key components (e.g., facilitators and matchmakers)
   - Remote monitoring (e.g., OAA Monitor agent)
   - Run-time environment for agent management (e.g., OAA Start-It)
   - Visualization services (e.g., RETSINA DemoDisplay agent)
5. Agent-based software designs allow system builders to take advantage of redundancy for system flexibility and robustness, by leveraging
   - Multiple agents with identical or similar capabilities
   - Multiple ways of entering requests
   - Multiple ways of getting things done

TIE 3 Components In addition to OAA infrastructure, and the OAA-RETSINA bridge agent, we contributed to TIE 3 our Multimodal Map system, weather and flight information retrieval agents, the Maestro video display agent Cheyer:IUI:1998, and telephone, speech recognition, and natural language understanding agents. The use of the Multimodal Map, phone, speech, and language agents is similar to that described for TIE 2, above. The weather and flight information retrieval agents answer queries about weather and airline schedules, by contacting publicly available Web sites. The Maestro agent is used in this TIE to display videos on screen (although it has a much broader range of capabilities).

Throughout the evolution of TIE 3, we made substantial contributions to scenario development, to the specification of interactions between the TIE agents, and to the development of the characterization of the TIE in terms of empirical hypotheses and experiments.

5.1.5 Moving the NEO TIEs onto the CoABS Grid

In the latter half of 1999, our NEO TIE efforts became focused on the need to adopt the CoABS Grid as the primary interoperability framework (and, at the same time, to contribute to the further evolution of the Grid). In collaboration with our TIE partners, we mapped out a plan for migrating the NEO TIE components onto the Grid. Because individually rewrapping all TIE components, as Grid components, was not feasible within given time constraints, we decided to pursue a (more interesting) strategy in which the three NEO TIE frameworks in use (OAA, RETSINA, and TEAMCORE) could continue to be employed, but could interoperate in limited ways through Grid services. We were able to draw on our experience from TIE 3 in building the OAA-RETSINA bridge agent.

Both the OAA Facilitator and the OAA-RETSINA bridge agent were retained in the Grid-enabled instantiation of the NEO TIEs. To provide access to OAA agents from the Grid, we designed and implemented a new Java-based *facilitator proxy agent*. The primary function of this agent was to translate OAA capabilities descriptions, at runtime, into Grid capabilities descriptions, and register them with the Grid registry component. In addition, we introduced the ability to log every message between OAA agents, using the Grid logging service.

Concurrently, we were active participants in several Grid affinity groups – those dealing with Middle Agents, Translation & Interoperation, and Ontology.

5.2 MIATA TIE

The objective of the MIATA TIE was to explore human/agent coordination issues for large-scale, continuous Command and Control. Specifically, the MIATA TIE focused on coordinating humans and agents from several military offices and nongovernmental organizations within a disaster relief domain. MIATA was grounded in a specific disaster scenario, namely, the passage of Hurricane Mitch through Honduras in 1998. The models underlying MIATA were drawn from actual historical data on the Hurricane Mitch relief effort.

The final demonstration system for MIATA showed users directing and interacting with agents to

- Form teams
- Gather intelligence about damage on the ground from a simulation of the region as the hurricane passes through
- Plan for the deployment of relief supplies and repair equipment
- Manage logistical resources and distribution of supplies
- React to problems on the ground

The TIE encompassed technologies from a wide range of contributors, including BBN Technologies, Carnegie Mellon University, the Kestrel Institute, the University of Rochester, the University of Oregon, SRI International, Yale, and OBJS. MIATA operates within a simulated testbed called Maplesim (`http://www.cs.cmu.edu/~maple/`).

5.2.1 Project Role within MIATA

Our role within the MIATA TIE was to demonstrate how agent directability technology could simplify management of a large number of agents within the highly dynamic operating environment of the disaster relief scenario. To this end, we developed a capability for intelligence gathering and dissemination, roughly analogous to the role of a J2. The information management task provided a good fit for demonstrating our technical ideas on human directability of agents, both because it requires a combination of goal-directed and reactive behavior, and because it is an information-rich task whose complexity is difficult for humans to manage on their own.

Our work for the MIATA TIE involved

- Formulating the specifics of the J2 role
- Working with the other TIE members to create a demonstration scenario
- Developing process descriptions for the J2 capability to manage simulated vehicles within CMU's Maplesim environment
- Implementing those process descriptions within PRS
- Integrating (both technically and conceptually) the J2 module with other MIATA components

To match the conceptualization of the MIATA framework, we implemented the J2 function in terms of a 'J2 proxy agent' that serves as an automated analog to the human J2. (There is a corresponding proxy agent for every other major functional role within MIATA.) This proxy agent accepts a variety of tasks related to the J2 function, and then manages the various agents under its control to accomplish those tasks. In addition, the proxy agent manages the communication of collected intelligence information to 'clients' that have requested it, or to other agents who need to be aware of it (e.g., repair crews who could respond to damaged bridges).

5.2.2 The TIGER System

Our software module within MIATA is called TIGER (TRAC Intelligence Gathering and Emergency Response). TIGER leverages a domain-independent implementation of our TRAC framework for agent guidance on top of PRS. TIGER runs on both Sun Workstations with Solaris, and PCs running Windows NT and 2000.

TIGER automates fully the processes required to fulfill the J2 function within the demonstration, and as such is capable of operating without human guidance. Our motivation, however, was to show that human directability of an agent community such as that within TIGER yields improved problem solving over agent systems where the human must either explicitly manage the agents himself, or has no control over the agents as they perform their assigned tasks.

TIGER interacted with virtually every other module in the MIATA system. As such, we spent a substantial amount of effort developing protocols and capabilities designed to fit the requirements of the overall MIATA framework. However, we designed TIGER in such a way that it can operate independently of all components in MIATA except the Maplesim simulator. This design enabled us to continue exploring our ideas on agent directability within TIGER after the completion of MIATA, independent of other MIATA participants.

TIGER Functionality TIGER provides the Intelligence Management component of a *disaster response team* whose objective is to provide humanitarian relief in the wake of a natural disaster. Other organizations within the team provide logistics (e.g., supplies distribution), operations (e.g., repair of infrastructure), and medical services. These organizations have their own physical assets (trucks and aircraft) available for their use. As would be expected, these organizations need to share information and resources to perform their functions effectively. A human commander oversees operations, dynamically tasking organizations to implement the relief process.

The primary role for TIGER is to gather information in response to requests from the supervisor or other members of the disaster response team. These requests can result in tasks to acquire information on the current state of infrastructure (roads, bridges) in designated regions, or to collect supply requirements (medical, food, water, shelter) of designated population centers within impacted regions. There can also be requests to be informed of key events (such as medical emergencies) as they become known. A secondary role is to respond to certain unexpected events (e.g., participating in evacuations, assisting with medical emergencies). Thus, TIGER agents must incorporate reactive capabilities that balance responsiveness with ongoing goal attainment.

The scope and complexity of the intelligence-gathering operations within the disaster relief context preclude step-by-step management of agent operations by a human. However, effective coordination of the available assets requires human supervision. As such, this domain provides an excellent example of an application that will benefit from technology for agent directability.

Agent Community Organization Figure 2 displays the organization of agents within TIGER.

The system has at its disposal a collection of simulated *physical agents* (trucks and helicopters) that can be used to gather information and respond to emergencies. This set may be changed by

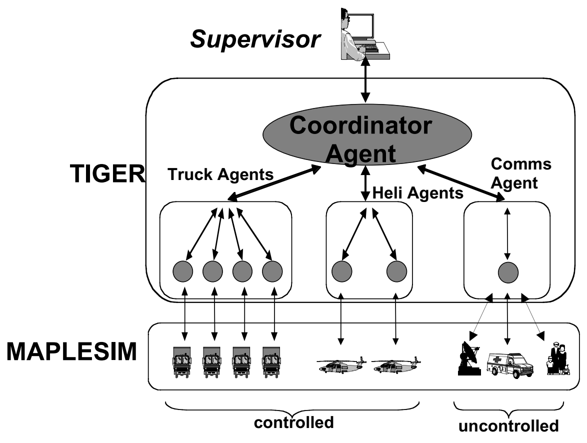


Figure 2: TIGER Architecture

external forces. For example, the logistics proxy agent may request to borrow some number of trucks to support delivery of critical supplies for a short period of time. Alternatively, when inundated with high-priority tasks that require immediate response, the J2 may request temporary access to resources controlled by other proxy agents. In addition, there is a set of simulated *communications agents* (e.g., other relief organizations, nongovernment organizations, local officials) that can be consulted to obtain information. TIGER contains a separate controller for each of the physical agents, as well as a communications manager for interacting with the various simulated communications agents. We refer to these controller agents as the *task execution agents* within TIGER, because they instigate and manage the activities required to perform assigned tasks.

The *coordinator agent* serves as the 'J2 proxy' within TIGER. It provides global management of tasks within the community, acting as a mediator between the human supervisor and the task execution agents. It also manages interactions with members of the disaster response team who request information (i.e., its *information clients*).

The TIGER agents communicate with other MIATA agents using KQML messages over the CoABS Grid, via the Grid proxy capability developed by BBN. Communications fall broadly into three categories:

- Task execution agents send messages to Maplesim to initiate and control simulated vehicles; they also receive status information from Maplesim corresponding to the percepts of the simulated vehicles.
- Communications agents send messages to Maplesim requesting information on specified topics from designated sources.
- The coordinator agent exchanges information with both human participants (e.g., human J2, J3, J4, and the commander) and agents from other modules within the disaster response team.[3]

[3]Figure 2 does not depict these additional MIATA modules.

Tasking Model Resource scheduling was not one of the aims of the TRAC project. For this reason, we opted to implement a simple heuristic task allocation algorithm within TIGER.

The TIGER coordinator agent maintains both a pool of unassigned tasks and a pool of currently unallocated agents. The coordinator agent matches a waiting task with an unallocated agent based on properties of the task, the available agents, and current knowledge about the state of the roads and bridges. Task properties include *location*, *priority* (an integer from 0 to 10), *type* (e.g., survey, rescue), and *status* (e.g., pending, completed, failed). The agent properties include *type* (e.g., helicopter or truck) and *location*.

Task management constitutes a major component of an execution agent's decision-making process. An execution agent must determine what to do if, while executing one task, the coordinator agent passes it a second task. It must also decide when to drop tasks that are not progressing well in favor of new tasks with higher potential for success.

For simplicity, we limit each task execution agent to at most one active task at any point in time. Agents may also have pending tasks (which they intend to undertake) and preempted tasks (which were begun but put aside for higher-priority tasks). Tasks are assigned to individual agents and do not require coordination with other agents for their completion.

Unexpected events (e.g., a medical emergency) may require immediate response. Events are characterized by the properties *location*, *time* (of the event), *severity* (an integer 0 to 10), *number of people affected*, and *type* (e.g., evacuation, medical). The coordinator agent selects an appropriate task execution agent to deal directly with each such event, thus bypassing the task pool.

These characteristics of tasking simplify the decision process for what an execution agent should do when it receives a task request. The agent can choose among several combinations of actions, including *ignore* the event, *adopt* a new task to respond to the event, *abandon* the current active task, *transfer* the task to another agent, or *postpone* the current task until the new task is completed. The agent's plan library includes options for each of these choices.

5.2.3 Contributions to MIATA

Within the final MIATA demonstration (January 2001), we showed how a human J2 can direct and adjust the behavior of the agents under its control within TIGER. In particular, the human J2 directed agents to interact with humans on decisions related to responding to certain types of emergency events, told agents not to deviate from certain survey tasks to respond to noncritical evacuations, and required agents always to respond to medical crises above certain severity thresholds. In this way, our agent directability technology enabled the human J2 to customize the behavior of his agents to meet the unique requirements of the demonstration scenario.

TIGER also allowed each client agent (human or automated) to customize the level of detail and timing of reports that it received. The client agent could request that reports on the status and/or results of tasks be generated periodically, or when particular sorts of events occurred (such as task execution problems or a medical emergency). This customized reporting provided a critical service within MIATA, as it enabled information clients to receive precisely the type and amount of information that they needed at any point in the scenario.

6 Conclusions

The TRAC framework for human directability of agents enables a user to define polices for adjustable agent autonomy and strategy preference. Using these mechanisms, a human supervisor can customize the operation of agents to suit his individual preferences and the dynamics of unexpected situations. In this way, system reliability and user confidence can be increased substantially over fully autonomous agent systems. The power of these ideas has been demonstrated within the TIGER system, which supports a human intelligence officer in managing a community of agents engaged in tasks for information gathering and emergency response.

Recognition of the need for technologies to support human-agent interactions has grown substantially in the past few years. However, few concrete technical approaches have been proposed to enable agent directability. Our work on policy-based agent directability defines a new and promising direction for enabling human controllability of agents.

Many outstanding issues in this area remain to be addressed. We briefly describe three important topics for future research.

Detecting and Resolving Guidance Conflicts Our conflict detection and resolution techniques provide a solid foundation for solving the general problem of robust response to conflicting guidance. However, much additional work is needed. First, techniques for dealing with indirect conflicts are required that reason about the downstream effects and requirements of plans. Second, our prioritization scheme for resolving direct conflicts presents a simple approach to conflicting guidance; it would be interesting to incorporate more advanced conflict resolution policies (e.g., in the style of [5, 9]). Finally, mixed-initiative resolution techniques will necessarily be a part of any effective conflict-handling facility.

Community Guidance The forms of agent directability considered in this project focus on influencing the behavior of an individual agent. Human supervisors will also want to express control at the *community* level, to encourage or discourage various forms of collective behaviors. The guidance "Keep 2 trucks within 15 miles of headquarters" is an example. Enforcement of this type of guidance will require mechanisms that support information exchange and coordinated action selection among groups of agents.

Collaborative Control Our model of agent directability provides a form of *supervised autonomy* [1] in which control over autonomy rests solely with the human supervisor. Some situations may benefit from a more collaborative approach [7], where both sides share control over initiative. For example, an agent may choose to initiate a dialogue with the human in situations where adherence to guidance would interfere with the pursuit of current goals, rather than blindly following the user's recommendations.

7 Bibliography

References

[1] K. S. Barber and C. E. Martin. Agent autonomy: Specification, measurement, and dynamic adjustment. In *Proceedings of the Autonomy Control Software Workshop at Autonomous Agents*, 1999.

[2] P. Bonasso. Issues in providing adjustable autonomy in the 3T architecture. In *Proceedings of the AAAI Spring Symposium on Agents with Adjustable Autonomy*, 1999.

[3] H. Chalupsky, Y. Gil, C. A. Knoblock, K. Lerman, J. Oh, D. Pynadath, T. A. Russ, and M. Tambe. Electric Elves: Applying agent technology to support human organizations. In *Proceedings of the Thirteenth Conference on Innovative Applications of Artificial Intelligence*, 2001.

[4] A. J. Cheyer and D. L. Martin. The Open Agent Architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, 4:143–148, 2001.

[5] F. Dignum, D. Morley, E. A. Sonenberg, and L. Cavedon. Towards socially sophisticated BDI agents. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS'2000)*, 2000.

[6] G. Ferguson and J. Allen. TRIPS: Towards a mixed-initiative planning assistant. In *Proceedings of the AIPS Workshop on Interactive and Collaborative Planning*, 1998.

[7] T. Fong, C. Thorpe, and C. Baur. Collaborative control: A robot-centric model for vehicle transportation. In *Proceedings of the AAAI Spring Symposium on Agents with Adjustable Autonomy*, 1999.

[8] M. P. Georgeff and F. F. Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

[9] E. Lupu and M. Sloman. Conflicts in policy-based distributed systems. *IEEE Transactions on Software Engineering, Special Issue on Inconsistency Management*, 25(6), 1999.

[10] D. L. Martin, A. J. Cheyer, and D. B. Moran. The Open Agent Architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13:91–128, 1999.

[11] D. B. Moran, A. J. Cheyer, L. E. Julia, D. L. Martin, and S. Park. Multimodal user interfaces in the Open Agent Architecture. In *Proceedings of the 1997 International Conference on Intelligent User Interfaces (IUI97)*, Orlando, FL, 6-9 January 1997.

[12] D. N. Morley. User's guide for the tiger system. Technical report, Artificial Intelligence Center, SRI International, 2002.

[13] K. L. Myers. Strategic advice for hierarchical planners. In L. C. Aiello, J. Doyle, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*. Morgan Kaufmann Publishers, 1996.

[14] K. L. Myers. Domain metatheories: Enabling user-centric planning. In *Proceedings of the AAAI-2000 Workshop on Representational Issues for Real-World Planning Systems*, 2000.

[15] K. L. Myers and D. N. Morley. Directing agent communities: An initial framework. In *Proceedings of the IJCAI Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*, 2001.

[16] K. L. Myers and D. N. Morley. Human directability of agents. In *Proceedings of the First International Conference on Knowledge Capture*, 2001.

[17] K. L. Myers and D. N. Morley. The TRAC framework for agent directability. Technical report, Artificial Intelligence Center, SRI International, 2001.

[18] K. L. Myers and D. N. Morley. Policy-based agent directability. In H. Hexmoor, R. Falcone, and C. Castelfranchi, editors, *Agent Autonomy*. Kluwer Academic Publishers, 2002.

[19] K. L. Myers and D. N. Morley. Resolving conflicts in agent guidance. In *Proceedings of the AAAI-2002 Workshop on Preferences in AI and CP: Symbolic Approaches*, 2002.

[20] K. L. Myers and D. N. Morley. Resolving conflicts in agent guidance. Technical report, Artificial Intelligence Center, SRI International, Menlo Park, CA, 2002.

[21] J. G. M. Paolucci and K. Sycara. Agent interoperation across multiagent system boundaries. In *Proceedings of the Fourth International Conference on Autonomous Agents (Agents 2000)*, New York, NY, June 2000. Association for Computing Machinery.

[22] A. S. Rao and M. P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*, San Francisco, 1995.

[23] D. Schreckenghost, J. Malin, C. Thronesbery, G. Watts, and L. Fleming. Adjustable control autonomy for anomaly response in space-based life support systems. In *Proceedings of the IJCAI Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*, 2001.

[24] K. Sycara, K. Decker, A. Pannu, and M. Williamson. Designing behaviors for information agents. In *Proceedings of the First International Conference on Autonomous Agents (AGENTS-97)*, February 1997.

[25] D. E. Wilkins and K. L. Myers. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation*, 5(6), 1995.

A Selected Publications

We produced the following technical papers and reports on the project; they are available at the project website `www.ai.sri.com/˜trac`.

- *Policy-based Agent Directability*, K. L. Myers and D. N. Morley. In *Agent Autonomy*, edited by H. Hexmoor, R. Falcone, and C. Castelfranchi, Kluwer Academic Publishers, 2002. †
- *Resolving Conflicts in Agent Guidance*, K. L. Myers and D. N. Morley. In *Proceedings of the AAAI-2002 Workshop on Preferences in AI and CP: Symbolic Approaches"*, Edmonton, Alberta, 2002. †
- *Human Directability of Agents*, K. L. Myers and D. N. Morley. In *Proceedings of the First International Conference on Knowledge Capture*, Victoria, B.C., 2001.
- *Directing Agent Communities: An Initial Framework*, K. L. Myers and D. N. Morley. In *Proceedings of the IJCAI Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*, Seattle, WA, 2001.
- *User's Guide for the TIGER System*, D. N. Morley. Technical Report, AI Center, SRI International, 2002.

To extend the technical summaries provided in the main body of this report, preliminary versions of the documents marked by † above are included in this appendix. (Final versions will be completed shortly for publication.)

Policy-based Agent Directability

Karen L. Myers, David N. Morley
Artificial Intelligence Center
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
myers@ai.sri.com, morley@ai.sri.com

Abstract

Many potential applications for agent technology require humans and agents to work together in order to achieve complex tasks effectively. In contrast, much of the work in the agents community to date has focused on technologies for fully autonomous agent systems. This paper presents a framework for the *directability* of agents, in which a human supervisor can define policies to influence agent activities at execution time. The framework focuses on the concepts of *adjustable autonomy* for agents (i.e., varying the degree to which agents make decisions without human intervention) and *strategy preference* (i.e., recommending how agents should accomplish assigned tasks). These mechanisms enable a human to customize the operations of agents to suit individual preferences and the dynamics of unexpected situations, leading to improved system reliability and increased user confidence over fully automated agent systems. The directability framework has been implemented within a PRS environment, and applied to a multiagent intelligence-gathering domain.

1 Introduction

The technical and public press are filled these days with visions of a not-too-distant future in which humans rely on software and hardware agents to assist with problem solving in environments both physical (e.g., smart offices, smart homes) and virtual (e.g., the Internet). The notion of *delegation* plays a central role in these visions, with humans off-loading responsibilities to agents that can perform activities in their place.

Successful delegation requires more than the mere assignment of tasks. A good manager generally provides directions to a subordinate so that tasks are performed to his or her liking. To ensure effectiveness, the manager will monitor the progress of the subordinates, occasionally interrupting to provide advice or to resolve problems.

The agents research community has, for the most part, focused on the mechanics of building autonomous agents and techniques for communication and coordination among agents. In contrast, little attention has been paid to supporting human interactions with agents of the type required for extended problem-solving sessions. Most agent frameworks lie at the extremes of the interaction spectrum, either assuming full automation by the agents with no means for user involvement, or requiring human intervention at each step along the way (i.e., *teleoperation* mode). Recently, however, there has been increased interest in agent systems designed specifically to support interaction with humans (e.g., [2, 3, 5, 14]).

This paper describes a framework, called Taskable Reactive Agent Communities (TRAC), that supports directability of a team of agents by a human supervisor. Within TRAC, the human assigns tasks to agents along with guidance that imposes boundaries on agent behavior. By adding, deleting, or modifying guidance at execution time, the human can manage agent activity at a level of involvement that suits his or her needs. In essence, our approach can be viewed as form of process management technology that enables flexible human control of agent communities.

A key issue in developing technology to support agent directability is determining the types of guidance to be provided. This paper focuses on guidance for *adjustable agent autonomy* and *strategy preferences*. Guidance for adjustable autonomy enables a supervisor to vary the degree to which agents can make decisions without human intervention. Guidance for strategy preferences constitutes recommendations on how agents should accomplish assigned tasks. Effective delegation and management by a human supervisor also requires visibility into ongoing agent operations. Although not described in this paper, the TRAC framework includes a capability for *customizable reporting* that enables a supervisor to tailor the amount, type, and frequency of information produced by agents to meet his evolving needs. Details can be found in [11].

The main contributions of this paper are the characterization of guidance for adjustable autonomy and strategy preference, presentation of a formal language for representing such guidance, the description of a semantic model for satisfaction of such guidance by an agent, and techniques for enforcing such guidance during agent operation.

The paper begins with a description of our underlying model for agents (Section 2). From there, we present an informal characterization of guidance for adjustable autonomy and strategy preferences (Section 3). Next, we describe a multiagent system, called TIGER, which instantiates the TRAC approach to directability for the application of multiagent intelligence gathering in the wake of a simulated natural disaster (Section 4). We use TIGER to provide examples of the directability concepts throughout this document. Following this description, we present our formal representation for guidance (Section 5) and a semantic model for guidance satisfaction (Section 6). Section 7 presents our techniques for guidance enforcement while Section 8 describes interface tools that support guidance specification. The paper concludes with a discussion of related work (Section 9).

2 Agent Model

We adopt a typical Belief-Desire-Intention (BDI) model of agency in the style of [12]. BDI agents are so-called due to the three components of their “mental state”: *beliefs* that the agent has about the state of the world, *desires* to be achieved, and *intentions* corresponding to plans of action that the agent has adopted to achieve its desires.

2.1 Agent Components

An agent represents the domain using a standard first-order language; *well-formed formulae* (*wffs*) are constructed from variables, quantifiers, connectives, and domain-specific predicate, function, and constant symbols.

The *beliefs* of an agent are represented by a set, *Bel*, of ground atomic facts. Given a set of beliefs, we define the truth of a wff ϕ (denoted by $Bel \models \phi$) as follows. A ground atomic fact, ϕ, is interpreted as true with respect to *Bel* iff $\phi \in Bel$ and false otherwise (i.e., the Closed World Assumption). The truth of a compound formula is derived from the truth of its constituents in the standard way.

The *desires* of an agent are represented by *goals* that denote conditions to be achieved. Syntactically, goals are constructed from goal name symbols and terms. As the BDI executor achieves a goal, it may bind variables in the goal, effectively returning values that result from goal achievement. Agents manipulate their world by performing *actions*. Syntactically, actions are constructed from action name symbols and ground terms.

Each agent has a library of *plans* that describe alternative ways of achieving a goal or of responding to a change in the belief state; our plan model is based on the Act representation language [15]. Plans are parameterized templates of activities that may require variable instantiations to apply to a particular situation. Each plan has a *precondition*, consisting of a wff stating conditions under which the plan can be used. The *cue* of a plan specifies a stimulus that activates the plan: *fact-invoked* plans have a wff for the cue and are triggered by the agent's beliefs changing to make that wff true; *goal-invoked* plans have a goal for the cue and are triggered by the posting of a unifying goal expression. The *body* of a plan specifies how to respond to the stimulus as a directed graph of actions to perform and subgoals to achieve.

An agent's plan library will generally contain a range of plans describing alternative responses to posted goals or events. Sets of these plans may be *operationally equivalent*, in that they share the same cue and preconditions but differ in the approach that they embody. To select among these alternatives, an agent may have some form of meta-control policy, such as user guidance.

A *plan instance* is a copy of a plan in the library with some substitution of terms for variables in the plan. A plan instance represents a possible way of responding to a triggering event. The *relevant* plans for an event (belief change or posted goal) are the plans in the library whose cue unifies with the event. The *applicable* plan instances for an event consist of instances of the relevant plans created by applying variable substitutions that unify the cue with the event, and cause the precondition to be true with respect to the agent's beliefs.

2.2 Agent Execution

The role of the BDI executor is to select plan instances to execute in response to changes in its beliefs and goals. An *intended plan* consists of a plan instance that the agent has decided to execute, together with information about the progress of that execution: the node in the body that is currently being executed and additional variable bindings that have been introduced through the execution. Intended plans exist as part of an *intention*, a hierarchical structure corresponding to an execution thread. The root of each intention consists of an intended plan that resulted from a belief change or from a goal supplied by the user. Down the intention hierarchy, the cue of each intended plan matches a goal in the intended plan above it. The *intention set*, *Int*, of a BDI agent is the set of all intentions that the agent is executing.

A BDI executor runs a continuous *sense-decide-act* loop. At the beginning of each cycle, the executor updates the agent's beliefs based on sensor information, and posts a belief change event for each change. Additionally, the executor posts a goal event for each new goal given to it by the user.

The executor then selects an intention and identifies the *current node* to be considered (either a goal or an action) in the body of the lowest-level intended plan of that intention. If the current node is an action, the action is attempted and any variable bindings that result from the successful execution of the action are applied to the intended plans of the selected intention. Otherwise, the current node is a goal and the agent posts a corresponding goal event.

For each posted event, the executor collects the applicable plan instances and selects one to be *intended* (i.e., for a goal posted from an intention, added to that intention; for other goals, creates a new intention). When the last goal or action of an intended plan body completes successfully, the intended plan is dropped from the intention, and any goal that triggered it is deemed completed.

The selection of an applicable plan instance to intend for an event is based on the *BDI executor state* that contains the *mental state* of the agent (i.e., the beliefs, desires, and intentions) along with the selected event. In this document, we focus on plan instance selection for a goal event, $\mathcal{G}^{ur}$, posted from an existing intention (for simplicity). Plan instance selection for belief change events and user specified goal events can be treated similarly. We denote a BDI executor state by the tuple $S = \langle Bel, Des, Int, \mathcal{G}^{ur} \rangle$.

Within this model of BDI execution, agents make four classes of decisions:

D1 *whether to respond to new goals and events*

D2 *how to select among multiple applicable plans when expanding a goal*

D3 *how to select instantiations for plan variables*.

D4 *which intention to consider*

3 TRAC Framework for Agent Directability

Our directability framework assumes that agents are capable of fully autonomous operation. More concretely, an agent's plan library covers the range of activities required to perform its assigned tasks. This assumption means that agents do not depend on the human supervisor to provide knowledge for task execution. Within this setting, guidance provides customization of agent behavior to suit the preferences of the human supervisor. In many applications, such guidance will enable superior performance, given that few plan libraries will reflect the full the experience, breadth of knowledge, and reasoning capabilities that a human supervisor can bring to the decision-making process.

Our model of agent directability focuses on general and task-specific policies to influence the activities undertaken by agents in their execution of assigned tasks. In particular, we emphasize the areas of (a) adjustable levels of agent autonomy and (b) strategy preferences that describe approaches to be used by an individual agent in executing assigned tasks.

3.1 Adjustable Autonomy

We define the autonomy of an agent to be the extent to which it is allowed to make decisions (specifically, D1 – D3) on its own. In situations where activities are routine and decisions straightforward, a human may be content to delegate all problem-solving responsibility to an agent. However, in situations where missteps could have severe consequences, the degree of autonomy of an individual agent should necessarily be controllable by a human.

We are interested in domains where agents will generally need to operate with high degrees of autonomy. For this reason, we assume a *permissive* environment: unless stated otherwise, agents are allowed to operate independent of human interaction. Our approach allows the human to adjust the scope of operations that can be undertaken by an agent on its own terms, focusing on the notions of *permission requirements* for action execution and *consultation requirements* for decision making.

Permission Requirements Permission requirements declare conditions under which an agent must elicit authorization from the human supervisor before executing actions. For example, the directive "Obtain permission before abandoning survey tasks with Priority $>$ 3" imposes the constraint that an agent request approval from the supervisor to abandon a certain class of tasks.

Consultation Requirements Consultation requirements designate a class of agent decisions that should be deferred to the human supervisor. These decisions can relate to either the selection of a value for variable instantiation (e.g., "Consult when selecting locations for staging bases") or the selection of a plan for a goal (e.g., "Consult when choosing a response to a failed survey task").

Our model of permission and consultation requirements, like earlier work on authority models, provides the means to block performance of certain actions by an agent. However, authority models are generally static (e.g., the *levels of autonomy* in [2]) and often derived from organizational structures. In contrast, our approach provides a rich language for expressing permission and consultation policies, which can vary throughout a problem-solving session.

3.2 Strategy Preference

Strategy preferences express recommendations on how an agent should accomplish tasks. These preferences could indicate specific plans to employ or restrictions on plans that should not be employed, as well as constraints on how plan variables can be instantiated.

For example, the directive "Try contacting Nongovernmental Organizations for information before sending vehicles to towns on the west coast" expresses a preference for selecting among operationally equivalent plans. On the other hand, the directive "Only use helicopters for survey tasks in sectors that are expected to be inaccessible by truck for more than 1 week" restricts the choice of resource type for instantiating certain plan variables.

4 The TIGER System

We have developed a prototype implementation of our TRAC framework for agent guidance on top of the Procedural Reasoning System (PRS) [6]. The TRAC implementation has been used as the basis for a demonstration system called TIGER (TRAC Intelligence Gathering and Emergency Response) that serves as a testbed for exploring our ideas on agent directability. Within TIGER, a human supervisor can delegate tasks to agents while providing guidance to control their runtime behavior.

4.1 TIGER Functionality

TIGER serves as part of a *disaster response team* whose objective is to provide humanitarian relief in the wake of a natural disaster. Other organizations within the team provide logistics (e.g., supplies distribution), operations (e.g., repair of infrastructure), and medical services. These organizations have their own physical assets (trucks and aircraft) available for their use. As would be expected, these organizations need to share information and resources to perform their functions effectively. A human commander oversees operations, dynamically tasking organizations to implement the relief process.[1]

The primary role for TIGER is to gather information in response to requests from the supervisor or other members of the disaster response team. These requests can result in tasks to acquire information on the current state of infrastructure (roads, bridges) in designated regions, or to collect supply requirements (medical, food, water, shelter) of designated population centers within impacted regions. There can also be

[1]The system operates within a testbed that simulates a major hurricane in Central America; the testbed is built on the MAPLE system (`http://www.cs.cmu.edu/~maple/`).

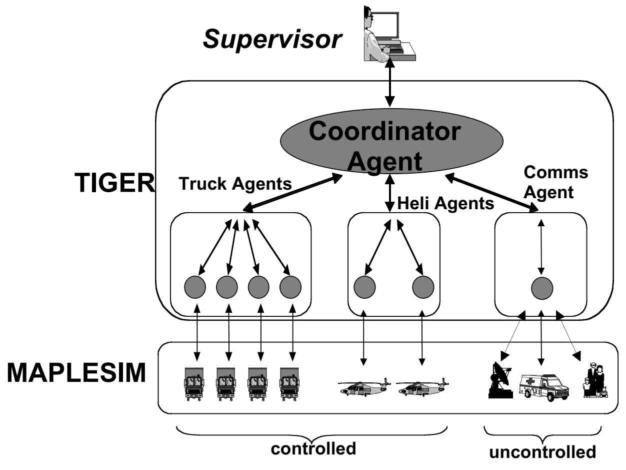


Figure 1: TIGER Architecture

requests to be informed of key events (such as medical emergencies) as they become known. A secondary role is to respond to certain unexpected events (e.g., participating in evacuations, assisting with medical emergencies). Thus, TIGER agents must incorporate reactive capabilities that balance responsiveness with ongoing goal attainment.

The scope and complexity of the intelligence-gathering operations within the disaster relief context preclude step-by-step management of agent operations by a human. However, effective coordination of the available assets requires human supervision. As such, this domain provides an excellent example of an application that will benefit from technology for agent directability.

4.2 Agent Community Organization

Figure 1 displays the organization of agents within TIGER. The system has at its disposal a collection of simulated *physical agents* (trucks and helicopters) that can be used to gather information and respond to emergencies. In addition, there is a set of simulated *communications agents* (other relief organizations, nongovernment organizations, local officials) that can be consulted to obtain information. TIGER contains a separate controller for each of the physical agents, as well as a communications manager for interacting with the various simulated communications agents. We refer to these controller agents as the *task execution agents* within TIGER, because they instigate and manage the activities required to perform assigned tasks.

The *coordinator agent* provides global management of tasks within the community, acting as a mediator between the human supervisor and the task execution agents. It also manages interactions with members of the disaster response team who request information (i.e., its *information clients*).

4.3 Tasking Model

The TIGER coordinator agent maintains both a pool of unassigned tasks and a pool of currently unallocated agents. The coordinator agent matches a waiting task with an unallocated agent based on properties of the task, the available agents, and current knowledge about the state of the roads and bridges. Task properties include *location*, *priority* (an integer from 0 to 10), *type* (e.g., survey, rescue), and *status* (e.g., pending, completed, failed). The agent properties include *type* (e.g., helicopter or truck) and *location*.

Task management constitutes a major component of an execution agent's decision-making process. An execution agent must determine what to do if, while executing one task, the coordinator agent passes it a second task. It must also decide when to drop tasks that are not progressing well in favor of new tasks with higher potential for success.

For simplicity, we limit each task execution agent to at most one active task at any point in time. Agents may also have pending tasks (which they intend to undertake) and preempted tasks (which were begun but put aside for higher-priority tasks). Tasks are assigned to individual agents and do not require coordination with other agents for their completion.

Unexpected events (e.g., a medical emergency) may require immediate response. Events are characterized by the properties *location*, *time* (of the event), *severity* (an integer 0 to 10), *number of people affected*, and *type* (e.g., evacuation, medical). The coordinator agent selects an appropriate task execution agent to deal directly with each such event, thus bypassing the task pool.

These characteristics of tasking simplify the decision process for what an execution agent should do when it receives a task request. The agent can choose among several combinations of actions, including *ignore* the event, *adopt* a new task to respond to the event, *abandon* the current active task, *transfer* the task to another agent, or *postpone* the current task until the new task is completed. The agent's plan library includes options for each of these choices.

5 Representation of Guidance

Our language for representing agent guidance builds on three main concepts: the underlying *agent domain theory*, a *domain metatheory*, and the connectives of first-order logic. Using these elements, we develop the main concepts underlying our model of agent guidance. These consist of an *activity specification* for describing abstract classes of action, a *desire specification* for describing abstract classes of goals, and an *agent context* for describing situations in which guidance applies.

5.1 Domain Metatheory

A standard domain theory for an agent consists of four types of basic element: *individuals* corresponding to real or abstract objects in the domain, *relations* that describe characteristics of the world, *goals* that an agent may adopt, and *plans* that describe available means for achieving goals.

The domain metatheory provides an abstracted characterization of elements of the domain theory that highlights key semantic differences. The role of the domain metatheory is to provide a high-level language for describing activity that abstracts from the details of an agent's internal representations. As discussed in [10], such a metatheory can provide a powerful basis for supporting user communication. The main concepts within our metatheory for agent guidance are *features* and *roles* (similar in spirit to those of [9]) defined for agent plans and goals.

Consider first plans. A *plan feature* designates an attribute of interest for a plan that distinguishes it from other plans that could be applied to the same task. For example, among plans for route determination, there may be one that is `OPTIMAL` but `SLOW` with a second that is `HEURISTIC` but `FAST`; each of these attributes could be modeled as a feature. Although the two plans are operationally equivalent (i.e., same cue and preconditions), their intrinsic characteristics differ significantly. Features provide the means to distinguish among such operationally equivalent alternatives.

A *plan role* describes a capacity in which a domain object is used within a plan; it maps to an individual variable within a plan. For instance, a route determination plan may contain variables `location.1` and `location.2`, with the former corresponding to the `START` and the latter the `DESTINATION`. Roles provide a semantic basis for describing the use of individuals within plans that abstracts from the details of specific variable names.

In analogous fashion, we can define goal features and goal roles. For example, a goal of informing another party of task progress may have a `COMMUNICATION` feature and `RECIPIENT` role associated with it. These metatheoretic elements can be used to specify the class of goals that involve communicating with the commander.

The value of the domain metatheory lies with its provision of a semantically motivated abstraction of the underlying planning domain. This abstraction builds on semantic linkage among domain elements, specifically the sharing of roles and features among multiple plans and goals. A domain metatheory would be developed in conjunction with the definition of the underlying domain theory for an agent. As discussed in [10], a domain metatheory should be a natural by-product of a principled approach to domain modeling.

We use the symbols f and r to denote features and roles. The symbols F^P and R^P denote the set of plan features and roles defined for a given agent; similarly, F^G and R^G denote the set of defined goal features and roles.

5.2 Activity and Desire Specifications

An *activity specification* characterizes an abstract class of plan instances for an agent. Our domain metatheory provides the basis for defining an activity specification, in terms of a set of required and prohibited features on a plan, and constraints on how plan roles are filled.

Definition 1 (Activity Specification) *An* activity specification $A = \langle F^+, F^-, R, \phi \rangle$ *consists of*

- *a set of* required features $F^+ \subseteq F^P$
- *a set of* prohibited features $F^- \subseteq F^P$,
- *a set of* roles $R = [r_1, \ldots, r_k] \subseteq R^P$ *and*
- *a* role-constraint formula $\phi[r_1, \ldots, r_k]$

For example, the following activity specification describes the class of plan instances with the feature `SURVEY` but not `HEURISTIC`, where the variables that fill the roles `START` and `DESTINATION` are instantiated to values in the same sector.

```
<{SURVEY},{HEURISTIC},{START, DESTINATION},{(= (SECTOR START) (SECTOR DESTINATION))}>
```

A *desire specification* constitutes the goal-oriented analogue of an activity specification, consisting of a collection of required and prohibited features from F^G, required roles from R^G, and role constraints. We use the symbol D to represent a generic desire specification.

5.3 Agent Context

Just as individual plans employ preconditions to limit their applicability, guidance rules require a similar mechanism for defining their scope. To this end, we introduce the notion of an *agent context*. While plan preconditions are generally limited to beliefs about the world state, our model of agent context focuses on the *BDI executor state* of an agent. As such, it is characterized in terms of the agent's beliefs, desires, intentions, as well as the current goal to which it is responding within a given cycle of the executor loop. Beliefs are specified in terms of constraints on the current world state. Desires are specified as desire specifications that describe goals that the agent has adopted, including the goal currently being expanded. Intentions are specified through activity specifications that describe intended plans being executed by the agent.

As discussed in Section 2, our model of agency assumes a hierarchical collection of plans and goals; furthermore, agents are capable of multitasking (i.e., executing multiple intentions in parallel). Within a given phase of the BDI execution cycle, goals for an agent of this type can be scoped in three ways:

- *Current goal:* the goal for which the BDI executor is selecting a plan
- *Local goals:* the current goal, or any of its ancestors
- *Global goals:* any goal of the agent

Distinguishing these different scopes for goals enables guidance to be localized to highly specialized situations. Plans under execution can be scoped similarly.

Definition 2 (Agent Context) *An* agent context *is defined by a tuple $C = \langle \Phi, D, A \rangle$, where*

- Φ *is a set of well-formed formulae*
- $D = D^C \cup D^L \cup D^G$ *is a set of current, local, and global desire specifications*
- $A = A^L \cup A^G$ *is a set of local and global activity specifications.*[2]

5.4 Permission Requirements

Permission requirements are defined in terms of an *agent context* and a *permission-constrained activity specification*. The agent context defines conditions on the operating state of the agent that limit the scope of the permission requirement. The permission-constrained activity specification designates a class of plan instances for which permission must be obtained.

Definition 3 (Permission Requirement) *A* permission requirement $\langle C, A \rangle$ *consists of an agent context C and an activity specification A.*

The interpretation of a permission requirement is that, when an agent's BDI state matches the specified agent context, permission must be obtained from the supervisor in order to execute a plan instance that matches the permission-constrained activity.

[2]The motivation for guidance is to influence the choice of plan to use for the current goal. Since there is not yet an intended plan for the current goal at the time when the decision is made, the agent context does not include activity specifications for the current plan.

Example 1 (Permission Requirement) The statement "Seek permission to abandon survey tasks with priority > 5" could be translated into a permission requirement of the form

```
Agent Context:
 Local Activity Spec:
  Features+: SURVEY
Permission-Constrained Activity Spec:
  Features+: ABANDON
  Roles: CURRENT-TASK
  Constraint: (> (TASK-PRIORITY CURRENT-TASK) 5)
```

5.5 Consultation Requirements

TRAC supports two types of consultation requirements: *value* and *plan*. A value consultation requirement consists of an *agent context* and a *consultation role*. The interpretation of a value consultation requirement is that when an agent's BDI executor state matches the agent context, any instantiation decision for a variable corresponding to the consultation role should be passed to the human supervisor. A plan consultation requirement consists of an *agent context* and a *desire specification*; it indicates that when an agent's BDI executor state matches the agent context, the human supervisor should be asked to select a plan to apply for any goal that matches the desire specification.

Definition 4 (Consultation Requirements) *A* value consultation requirement $\langle C, r\rangle$ *consists of an agent context* C *and a role* r*. A* plan consultation requirement $\langle C, D\rangle$ *consists of an agent context* C *and a desire specification* D*.*

Example 2 (Consultation Requirements) The guidance "When responding to medical emergencies, consult when selecting a medical evacuation site" could be translated into a consultation requirement of the form

```
Agent Context:
 Local Activity Spec:
  Features+: RESPONSE, MEDICAL-EMERGENCY
 Consultation Role: MEDEVAC-SITE
```

The guidance "Consult when choosing a response to a failed survey task" could be translated into a plan consultation requirement of the form

```
Agent Context:
 Local Activity Spec:
  Features+: SURVEY, FAILURE
 Current Desire Spec:
  Features+: RESPONSE
```

5.6 Strategy Preference

Strategy preference guidance consists of two components: an *agent context* and a *response activity specification*. The activity specification designates the class of recommended plan instances to be applied (i.e., choice of plan and variable instantiations for designated roles) when an agent's state that matches the designated agent context.

Definition 5 (Strategy Preference) *A* strategy preference *rule is defined by a pair* $\langle C, A\rangle$ *where* C *is an agent context and* A *is an activity specification.*

Example 3 The statement "Don't take on medical emergencies involving fewer than 5 people when the current task priority exceeds the emergency severity" could be represented by the following strategy preference:

```
Agent Context:
 Belief: (CURRENT-TASK task.1)
 Current Desire Spec:
  Features+: RESPONSE
  Roles: EVENT
  Constraint:
   (AND (= (EVENT-TYPE EVENT) MEDICAL-EMERGENCY)
        (< (EVENT-NUMBER-AFFECTED EVENT) 5)
        (> (TASK-PRIORITY CURRENT-TASK) (EVENT-SEVERITY EVENT)))
Response Activity Spec:
  Features-: ADOPT
```

A goal with the feature `RESPONSE` and role `EVENT` triggers consideration of the guidance, provided that `EVENT` has type `MEDICAL-EMERGENCY` with fewer than 5 affected people, and the priority of `CURRENT-TASK` is greater than the severity of `EVENT`. The response activity specification indicates not to adopt responsibility for the emergency in such cases.

6 Guidance Semantics

Semantically, guidance acts as a filter on the plan instances that an agent can execute. When a standard BDI agent attempts to find an instance of a plan from its library to apply to a goal, it determines a set of applicable plan instances based on the plan cues and preconditions. The guidance limits this set further in accord with the following conventions.

A guidance rule is deemed *relevant* iff its agent context matches the current BDI executor state of the agent. Each relevant strategy preference rule filters out plan instances that do not match the response activity specification. Each relevant permission requirement rule filters out plan instances that match the permission-constrained activity specification but are refused permission by the supervisor. Each relevant value consultation rule filters out plan instances that have the consultation role but do not bind the corresponding role variable to a value desired by the supervisor. Each relevant plan consultation rule filters out all plan instances other than that selected by the supervisor.

This section defines the semantics of guidance more formally. Section 6.1 defines matching for activity specifications, desire specifications, and agent context; these concepts are used in Section 6.2 to define guidance satisfaction.

6.1 Matching

Let a be either a plan or a goal. The function $\mathit{Features}(a)$ designates the features defined for a while $\mathit{Roles}(a)$ designates the roles defined for a. The function $\mathit{RoleVal}(a,r)$ designates the term that instantiates the role r in a (if one exists). For an uninstantiated plan p, $\mathit{RoleVal}(p,r)$ will generally be a variable. For a plan instance, $\mathit{RoleVal}(p,r)$ reflects any bindings for the role. To simplify the presentation, we restrict the structure of plans so that roles are bound to ground terms as part of the testing of applicability of a plan, through unification with the cue and/or precondition testing. We use the notation $\phi[x_1 : v_2, \ldots\ x_n : v_n]$ to represent a well-formed formula ϕ in which each occurrence of the variable x_i is replaced by the value v_i.

Definition 6 (Activity Specification Match) *A plan instance p matches an activity specification* $\mathcal{A} = \langle \mathcal{F}^+, \mathcal{F}^-, \mathcal{R}, \phi \rangle$ *in BDI executor state* $S = \langle \mathit{Bel}, \mathit{Des}, \mathit{Int}, g^{cur} \rangle$ *iff:*

- $\mathit{Features}(p) \subseteq \mathcal{F}^+$
- $\mathcal{F}^- \cap \mathit{Features}(p) = \emptyset$
- $\mathcal{R} \subseteq \mathit{Roles}(p)$
- $\mathit{Bel} \models \phi[r_1 : \mathit{RoleVal}(p,r_1) \ldots r_k : \mathit{RoleVal}(p,r_1)]$

Desire specification matches are defined similarly. We use the notation $\mathit{ActivityMatch}(p,\mathcal{A},S)$ $\mathit{DesireMatch}(g,\mathcal{D},S)$ to denote activity specification and desire specification matches, respectively.

Let $\mathit{Intention}(g)$ be the intention that produced goal g, let $\mathit{AllPlans}(i)$ be the plan instances chosen for execution within intention i, and let $\mathit{AllGoals}(i)$ be the current goals of those plan instances.

Definition 7 (Agent Context Match) *A BDI executor state* $S = \langle \mathit{Bel}, \mathit{Des}, \mathit{Int}, g^{cur} \rangle$ *matches an agent context* $\mathcal{C} = \langle \mathcal{B}, \mathcal{D}, \mathcal{A} \rangle$ *iff:*

- *For all* $\phi \in \mathcal{B}$, $\mathit{Bel} \models \phi$,
- *For all* $\mathcal{D} \in \mathcal{D}^C$, $\mathit{DesireMatch}(g^{cur}, \mathcal{D}, S)$,
- *For all* $\mathcal{D} \in \mathcal{D}^L$, $\exists g \in \mathit{AllGoals}(\mathit{Intention}(g^{cur}))$. $\mathit{DesireMatch}(g,\mathcal{D},S)$,
- *For all* $\mathcal{D} \in \mathcal{D}^G$, $\exists i \in \mathit{Int}, g \in \mathit{AllGoals}(i)$. $\mathit{DesireMatch}(g,\mathcal{D},S)$,
- *For all* $\mathcal{A} \in \mathcal{A}^L$, $\exists p \in \mathit{AllPlans}(\mathit{Intention}(g^{cur}))$. $\mathit{ActivityMatch}(p,\mathcal{A},S)$,
- *For all* $\mathcal{A} \in \mathcal{A}^G$, $\exists i \in \mathit{Int}, p \in \mathit{AllPlans}(i)$. $\mathit{ActivityMatch}(p,\mathcal{A},S)$

We use the notation $\mathit{ContextMatch}(\mathcal{C},S)$ to denote that an agent context $\mathcal{C}$ matches a BDI executor state S.

6.2 Guidance Satisfaction

Building on the definitions from the previous section, we define the concept of *satisfaction* of agent guidance. In the following definitions, we distinguish *trivial* from *nontrivial* satisfaction. Trivial satisfaction occurs when the guidance rule doesn't impact the selection of plan instance for a given executor cycle. This can occur, for example, because the guidance is not *relevant* in the current state.

Definition 8 (Satisfaction: Strategy Selection Rule) *A plan instance p* trivially satisfies *a strategy selection rule* $R_S = \langle \mathcal{C}, \mathcal{A} \rangle$ *for a BDI executor state S iff* $\mathit{ContextMatch}(\mathcal{C},S)$ *does not hold; p* nontrivially satisfies R_S *iff* $\mathit{ActivityMatch}(p,\mathcal{A},S)$ *holds.*

The guidance rules for permission and consultation may require explicit feedback from the human supervisor. We model this interaction using the following *oracle fluents*:

- $Permission(p,S)$ specifies whether the human supervisor authorizes an agent with BDI executor state S to adopt plan p
- $ValueChoice(r,V,S)$ specifies the human supervisor's preferred instantiation for role r among values in V, in BDI executor state S
- $PlanChoice(D,P,S)$ specifies the human supervisor's preferred plan in P for goals that match D, in BDI executor state S.

Definition 9 (Satisfaction: Permission Rule) *A plan instance p* trivially satisfies *a permission requirement rule $R_P = \langle C,A\rangle$ for a BDI executor state S iff either of $ContextMatch(C,S)$ or $ActivityMatch(p,A,S)$ does not hold; p* nontrivially satisfies *R_P iff $Permission(p,S) = True$ holds.*

Whereas permission requirements relate to *individual* plans, consultation requirements apply when there are choices among role values or applicable plans. Thus, the decision on whether consultation is necessary depends on what other plans are applicable.

Definition 10 (Satisfaction: Value Consultation Rule) *Let $R_{VC} = \langle C,r\rangle$ be a value consultation rule, $S = \langle Bel, Des, Int, g^{cur}\rangle$ be a BDI executor state, and P the set of applicable plans for g^{cur}. Let V be the set of instances to which r is instantiated in P, i.e., $V = \{RoleVal(p,r) \mid p \in P \wedge r \in Roles(p)\}$. A plan $p \in P$* trivially satisfies *R_{VC} for S iff either $ContextMatch(C,S)$ does not hold, $r \notin Roles(p)$, or V contains one or fewer elements; p* nontrivially satisfies *R_{VC} for S iff $RoleVal(p,r) = ValueChoice(r,V,S)$.*

Definition 11 (Satisfaction: Plan Consultation Rule) *Let $R_{PC} = \langle C,D\rangle$ be a plan consultation rule and $S = \langle Bel, Des, Int, g^{cur}\rangle$ a BDI executor state. Let P be the set of applicable plans for g^{cur}. A plan $p \in P$* trivially satisfies *R_{PC} for S iff either $ContextMatch(C,S)$ does not hold, $DesireMatch(g,D,S)$ does not hold, or P contains one or fewer elements; a plan p* nontrivially satisfies *R_{PC} for S iff $p = PlanChoice(D,P,S)$.*

We say that an agent *violates* a piece of guidance during a given executor loop cycle iff the agent selects a plan instance for the current goal that does not satisfy the guidance. In the ideal, an agent executor would avoid violating any user-provided guidance during its operation. However, factors beyond the executor's control will generally make it impossible to avoid all such violations. In particular, users may provide *conflicting guidance* to an agent that recommends incompatible choices. Conflicts can arise in different forms. Here, we distinguish between *direct* and *indirect* conflicts.

Direct conflicts arise when strategy preference guidance yields inconsistent recommendations within a given BDI executor cycle. Such conflicts can be at the level of plan instances (e.g., *Execute P* and *Don't execute P*) or the level of variable bindings (e.g., *Instantiate role R to A* and *Instantiate role R to B*).

Indirect conflicts arise when guidance recommends multiple plan instances for execution such that, while their execution can be initiated, it is impossible for all of them to complete successfully. For example, the simultaneous execution of two plan instances could lead to deadlock or livelock situations, or downstream resource contention. Such harmful interactions can arise within any multithreaded system, not just systems in which guidance is used to select activities. Because general mechanisms for detecting these interactions do not yet exist, we consider only direct conflicts in the remainder of this paper.

Given the potential for conflicting guidance, the best that we can expect from an executor is that it satisfy as much guidance as possible with each plan selection decision. The following definition of *maximally guidance compliant* captures this requirement.

Definition 12 (Maximally Guidance Compliant) *An executor is called* maximally guidance compliant *iff for a given set of guidance rules G, the executor selects a plan instance p that satisfies a maximal subset of G. That is, if the executor selects a plan instance p such that p satisfies* G^+ *and violates* G^- *where* $G = G^+ \cup G^-$*, then there is no applicable plan instance* p' *that satisfies* $G^+ \cup \{R\}$ *for any* $R \in G^-$*.*

7 Enforcement of Guidance

In this section, we describe a simple extension to the BDI executor from Section 2 that ensures maximal guidance compliance for a set of strategy selection, permission requirement, and consultation requirement guidance rules. This set can vary over time but is assumed fixed for a given iteration of the executor loop.

Enforcement of guidance is attained through a simple modification to the executor loop at the point where a plan instance is selected to be intended in response to a posted goal or fact. First, the current BDI executor state for an agent is matched to the agent context of all current guidance to determine the relevant guidance for the current execution cycle. Each piece of relevant guidance is then evaluated to determine which applicable plan instances lead to violations, with each plan instance being tagged with the guidance that it violates. A strategy preference rule will eliminate plan instances that do not match their response activity specification. For a permission rule, the human supervisor is queried to determine whether the plan instance is allowed. For a consultation rule, the user is queried for any plan or role instantiation choices.

If after consideration of all relevant guidance rules there remains one or more plan instances with no violations, then any of them can be selected for application. If every plan instance has at least one violation, then the executor selects a piece of guidance and eliminates the violations associated with it. Any plan instance without violations would then be considered for application. This process repeats until at least one such plan instance results. The agent then applies the selected plan instance to the current goal.

Different selection strategies can be adopted for deciding the order in which to drop guidance rules. One obvious strategy would prefer permission and consultation requirements over strategy selection rules, since the former incorporate situation-specific information regarding user preferences. Additionally, one could define *weights* that reflect relative strength of preference for guidance rules. A policy for combining and comparing the weights associated with the guidance rules that made the conflicting recommendations can then be used to select guidance to ignore, as a way of eliminating the conflict. TIGER incorporates this type of approach to deal with direct conflicts.

We refer to the above BDI executor algorithm as the *guidance filtering executor*. It is straightforward to establish the following proposition.

Proposition 1 (Guidance Compliance) *The guidance filtering executor is maximally guidance compliant.*

8 Guidance Interface Tools

The motivation for our work on agent directability is to enable users to direct and manage agents in dynamic, unpredictable environments. The language presented in earlier sections provides a highly expressive formalism in which to define agent guidance; however, the complexity of the language could overwhelm a typical user. For this reason, we have developed two tools to help users define and manipulate agent guidance.

The first tool is a *guidance authoring interface* that walks the user through the process of constructing a complex piece of guidance. To enable a simple specification process, the tool does not support the full expressivity of the formal guidance language; however, it supports a broad range of expressions, including the examples described in this paper. An accompanying *guidance library* can be used to store authored guidance. Users can select guidance from the library, as appropriate for a particular situation.

The second tool is a *permissions window* that enables users to activate and deactivate permission requirements for certain classes of action performed on certain types of task. In particular, selections made through this interface are compiled into corresponding permission requirement structures. While this interface limits the scope of permission requirements that can be expressed, it provides a simple, accessible specification mechanism.

9 Related Work

Recognition of the need for technologies to support human-agent interactions has grown substantially in the past few years. However, few concrete technical approaches have been proposed to enable agent directability.

Scerri et al. [13] apply Markov decision processes (MDPs) to provide a form of adjustable agent autonomy. Their approach involves defining an MDP that describes all possible courses of action. The agent uses expected utility estimates from this model to determine when to consult the supervisor, and adjusts the model parameters based on experience. To avoid learning inappropriate behavior, users can impose constraints on what can be learned. In contrast to our approach of having a human explicitly define a policy for autonomy, an agent within this framework determines an appropriate level on its own.

Schreckenghost et al. [14] apply the concept of adjustable autonomy to the management of space-based life support systems. In their system, a human can take over both the selection of tasks to perform and the execution of those tasks. In contrast to our use of explicit policies, the level of autonomy is specified by directly altering a "level of autonomy" setting (*manual* vs. *autonomous*) for all tasks, for a subsystem, or for an individual task.

Our strategy preference guidance selects among previously defined alternative plans; it does not expand the behavioral capabilities of the agent. In contrast, the work on *policy-based control* for distributed systems management supports runtime definition of new behaviors (e.g., [8]). Policy languages in this area focus on the concepts of *authority* and *obligation* to perform actions.

10 Conclusion

This paper presents a framework for human directability of agents that enables a user to define polices for adjustable agent autonomy and strategy preference. Through these mechanisms, a human supervisor can customize the operation of agents to suit his individual preferences and the dynamics of unexpected situations. In this way, system reliability and user confidence can be increased substantially over fully autonomous agent systems. The power of these ideas has been demonstrated within the TIGER system, which supports a human intelligence officer in managing a community of agents engaged in tasks for information gathering and emergency response.

Many outstanding issues in this area remain to be addressed; we briefly describe three topics for future work.

Detecting and Resolving Guidance Conflicts As discussed above, TIGER recognizes only a limited class of guidance-related conflicts (namely, direct conflicts among guidance). Indirect conflicts among guidance, and conflicts between guidance and ongoing activities require more powerful detection methods that reason about the downstream effects and requirements of plans. Furthermore, our prioritization scheme for resolving direct conflicts presents a simple approach to conflicting guidance; it would be interesting to incorporate more advanced conflict resolution policies (e.g., [4, 7]).

Community Guidance The forms of agent directability described in this paper focus on influencing the behavior of an individual agent. Human supervisors will also want to express control at the *community* level, to encourage or discourage various forms of collective behaviors. The guidance "Keep 2 trucks within 15 miles of headquarters" provides an example. Enforcement of this type of guidance will require mechanisms that support information exchange and coordinated action selection among groups of agents.

Collaborative Control Our model of agent directability provides a form of *supervised autonomy* [1] in which control over autonomy rests solely with the human supervisor. Some situations may benefit from a more collaborative approach , where both sides share control over initiative. For example, an agent may choose to initiate a dialogue with the human in situations where adherence to guidance would interfere with the pursuit of current goals, rather than blindly following the user's recommendations.

11 Acknowledgments

The authors thank Eric Hsu for his contributions in developing the TIGER interface, and Sebastian Thrun and his group at CMU for providing the MAPLE simulator. This work was supported by DARPA under the supervision of Air Force Research Laboratory contract F30602-98-C-0160.

References

[1] K. S. Barber and C. E. Martin. Agent autonomy: Specification, measurement, and dynamic adjustment. In *Proceedings of the Autonomy Control Software Workshop at Autonomous Agents*, 1999.

[2] P. Bonasso. Issues in providing adjustable autonomy in the 3T architecture. In *Proceedings of the AAAI Spring Symposium on Agents with Adjustable Autonomy*, 1999.

[3] H. Chalupsky, Y. Gil, C. A. Knoblock, K. Lerman, J. Oh, D. Pynadath, T. A. Russ, and M. Tambe. Electric Elves: Applying agent technology to support human organizations. In *Proceedings of the Thirteenth Conference on Innovative Applications of Artificial Intelligence*, 2001.

[4] F. Dignum, D. Morley, E. A. Sonenberg, and L. Cavedon. Towards socially sophisticated BDI agents. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS'2000)*, 2000.

[5] G. Ferguson and J. Allen. TRIPS: Towards a mixed-initiative planning assistant. In *Proceedings of the AIPS Workshop on Interactive and Collaborative Planning*, 1998.

[6] M. P. Georgeff and F. F. Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

[7] E. Lupu and M. Sloman. Conflicts in policy-based distributed systems. *IEEE Transactions on Software Engineering, Special Issue on Inconsistency Management*, 25(6), 1999.

[8] J. D. Moffett and M. S. Sloman. Policy hierarchies for distributed systems management. *IEEE Journal on Selected Areas in Communications*, 11(9), 1993.

[9] K. L. Myers. Strategic advice for hierarchical planners. In L. C. Aiello, J. Doyle, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*. Morgan Kaufmann Publishers, 1996.

[10] K. L. Myers. Domain metatheories: Enabling user-centric planning. In *Proceedings of the AAAI-2000 Workshop on Representational Issues for Real-World Planning Systems*, 2000.

[11] K. L. Myers and D. N. Morley. Directing agent communities: An initial framework. In *Proceedings of the IJCAI Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*, 2001.

[12] A. S. Rao and M. P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*, San Francisco, USA, 1995.

[13] P. Scerri, D. Pynadath, and M. Tambe. Adjustable autonomy in real-world multi-agent environments. In *Proceedings of the International Conference on Autonomous Agents*, 2001.

[14] D. Schreckenghost, J. Malin, C. Thronesbery, G. Watts, and L. Fleming. Adjustable control autonomy for anomaly response in space-based life support systems. In *Proceedings of the IJCAI Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*, 2001.

[15] D. E. Wilkins and K. L. Myers. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation*, 5(6), 1995.

Resolving Conflicts in Agent Guidance

Karen L. Myers   David N. Morley
Artificial Intelligence Center
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
myers@ai.sri.com   morley@ai.sri.com

Abstract

For agent technology to be accepted in real-world applications, humans must be able to customize and control agent operations. One approach for providing such controllability is to enable a human supervisor to define *guidance* for agents in the form of policies that establish boundaries on agent behavior. This paper considers the problem of conflicting guidance for agents, making contributions in two areas: (a) outlining a space of conflict types, and (b) defining resolution methods that provide robust agent operation in the face of conflicts. These resolution methods combine a guidance-based preference relation over plan choices with an ability to extend the set of options considered by an agent when conflicts arise. The paper also describes a PRS-based guidance conflict-handling capability applied within a multiagent intelligence-gathering domain.

Introduction

Many potential applications for agent technology require that humans and agents work together in order to accomplish tasks effectively. This requirement is especially important for domains where task complexity precludes formalization of agent behaviors for all possible eventualities. In such domains, the availability of mechanisms by which a human supervisor can provide direction will enable agents to be informed by the experience, breadth of knowledge, and superior reasoning capabilities that a human expert can bring to the problem-solving process.

In previous work, we defined a framework for *agent guidance* that supports dynamic directability of agents by a human supervisor (Myers & Morley 2001; 2002). Guidance imposes boundaries on agent behavior, thus enabling a human to customize and direct agent operations to suit his or her individual requirements. The guidance framework focuses on two types of agent directability, namely *adjustable agent autonomy* and *strategy preferences*. Guidance for adjustable autonomy enables a supervisor to vary the degree to which agents can make decisions without human intervention. Guidance for strategy preferences constitutes recommendations on how agents should accomplish assigned tasks. For example, the directive "Use helicopters for survey tasks in sectors on the west coast" imposes restrictions on how resources can be used to perform a certain class of task.

User guidance provides a powerful mechanism for run-time customization of agent behavior. However, it also introduces the potential for problems in the event that the guidance recommends inconsistent responses. Such conflicts cannot arise with adjustable autonomy guidance, but are a significant issue for strategy preference guidance. Robustness of operations requires mechanisms for detecting these conflicts and responding in a manner that does not jeopardize agent stability.

This paper addresses two main issues related to the topic of conflicting strategy preference guidance for agents. First, it identifies different types of conflict that can arise. Second, it defines automated techniques for resolving the conflicts. Our approach combines the selective dropping of problematic pieces of guidance with a proactive capability to eliminate the source of conflicts by modifying current agent activities. We do not consider interactive techniques for conflict resolution in this paper, although they would certainly play an important role in a comprehensive conflict resolution system.

The conflict resolution techniques defined in this paper have been implemented within the Taskable Reactive Agent Communities (TRAC) framework (Myers & Morley 2001; 2002), which provides a domain-independent guidance capability for PRS agents (Georgeff & Ingrand 1989). The techniques have been applied within the context of a simulated disaster relief task force in which a human supervisor must manage a team of agents engaged in a variety of information-gathering and emergency response tasks. Within this testbed, referred to as TIGER (TRAC Intelligence Gathering and Emergency Response), agents control simulated trucks and helicopters in the performance of their assigned tasks.

We begin with a description of the agent model that underlies our work on agent guidance, followed by a definition of strategy preference guidance. Next, we outline different types of conflict that can arise with agent guidance and our conflict resolution methods. Finally, we describe a realization of the conflict resolution mechanisms within the TIGER framework and discuss related work.

Agent Model

We adopt a typical Belief-Desire-Intention (BDI) model of agency in the style of (Rao & Georgeff 1995). BDI agents are so-called due to the three components of their "mental state": *beliefs* that the agent has about the state of the world, *desires* to be achieved, and *intentions* corresponding to actions that the agent has adopted to achieve its desires.

Each agent has a library of *plans* that defines the range of activities that an agent can perform to respond to events or to achieve assigned tasks; our plan model is based on (Wilkins & Myers 1995). Plans are parameterized templates of activities that may require variable instantiations to apply to a particular situation. In a standard BDI framework, there are two types of plan: *fact-invoked* plans for responding to changes in the beliefs of the agent, and *goal-invoked* plans for decomposing tasks into constituent subgoals and actions.

Each plan has a *cue* that specifies a stimulus that activates the plan, either a new goal (for a goal-invoked plan) or a change in the agent's beliefs (for a fact-invoked plan). A set of *preconditions* associated with plans defines gating constraints that must be satisfied for a plan to be applied. A plan is said to be *relevant* to a world change (e.g., new goal or belief change event) if the plan cue matches the stimulus, and *applicable* if, additionally, the plan preconditions are satisfied relative to the agent's current beliefs. The *body* of a plan specifies how to respond to the stimulus, in terms of actions to perform and subgoals to achieve.

An agent's plan library will generally contain a range of plans describing alternative responses to posted goals or events. Sets of these plans may be *operationally equivalent* (i.e., they share the same cue and preconditions) but differ in the approach that they embody. Some form of metacontrol policy can be defined to select among such alternatives, such as strategy preference guidance.

A BDI interpreter runs a continuous *sense-decide-act* loop to respond to changes in its operating environment. At the start of each cycle, the interpreter collects all new goals and events (i.e., changes in its beliefs about the world). Next, it determines whether there are any plans in its library that are applicable to these changes. From this set, it selects some subset for execution and creates intentions for them. Finally, some bounded number of steps for each current intention are performed.

The guidance framework assumes that agents are capable of fully autonomous operation. More concretely, an agent's plan library covers the range of activities required to perform its assigned tasks. This assumption means that agents do not depend on the human supervisor to provide knowledge to complete tasks. Within this setting, guidance provides customization of agent behavior to suit the preferences of the human supervisor. In many applications, such guidance will enable superior performance, given that few plan libraries will reflect the experience, breadth of knowledge, and reasoning capabilities that a human supervisor can bring to the decision-making process.

Strategy Preference Guidance

Strategy preference guidance expresses recommendations on how an agent should accomplish tasks. These preferences could designate classes of plans to employ or restrictions on plans that should not be employed, as well as constraints on how plan variables can be instantiated. For example, the directive "Try contacting Nongovernmental Organizations for information before sending vehicles to towns on the west coast" expresses a preference for selecting among operationally equivalent plans. The directive "Only use helicopters for survey tasks in sectors that are expected to be inaccessible by truck for more than 1 week" restricts the choice of resource type for instantiating certain plan variables.

Representation of Guidance

Our language for representing agent guidance builds on three main concepts: the underlying *agent domain theory*, a *domain metatheory*, and the connectives of first-order logic. Using these elements, we develop the main concepts underlying our model of agent guidance. These consist of an *activity specification* for describing abstract classes of action, a *desire specification* for describing abstract classes of goal, and an *agent context* for describing situations in which guidance applies.

Domain Metatheory A standard domain theory for an agent consists of four types of basic element: *individuals* corresponding to real or abstract objects in the domain, *relations* that describe characteristics of the world, *goals* that an agent may adopt, and *plans* that describe available means for achieving goals.

The *domain metatheory* provides an abstracted characterization of elements of the domain theory that highlights key semantic differences. As discussed in (Myers 2000a), a metatheory can yield a rich vocabulary for describing activity, thus providing a powerful basis for supporting user communication. The main concepts within our metatheory for agent guidance are *features* and *roles* (similar in spirit to those of (Myers 1996)) defined for agent plans and goals.

Consider first plans. A *plan feature* designates an attribute of interest for a plan that distinguishes it from other plans that could be applied to the same task. For example, among plans for route determination, there may be one that is *Optimal* but *Slow* with a second that is *Heuristic* but *Fast*; each of these attributes could be modeled as a feature. Although the two plans are operationally equivalent (i.e., same cue and preconditions), their intrinsic characteristics differ significantly. Features provide the means to distinguish between such operationally equivalent alternatives.

A *plan role* describes a capacity in which a domain object is used within a plan; it maps to an individual variable within a plan. For instance, a route determination plan may contain variables *location.1* and *location.2,* with the former corresponding to the *Start* and the latter the *Destination*.

In analogous fashion, roles and features can also be defined for goals. For example, a goal of informing another party of task progress may have a *Communication* feature and *Recipient* role associated with it. These metatheoretic constructs can be used to specify the class of goals that involve communicating with the commander.

Activity and Desire Specification An *activity specification* characterizes an abstract class of plan instances for an agent. Our domain metatheory provides the basis for defining an activity specification, in terms of a set of required and prohibited features on a plan, as well as constraints on the way in which plan roles are filled.

Definition 1 (Activity Specification) *An* activity specification $\alpha = \langle F^+, F^-, R, \phi \rangle$ *consists of*

- *a set of* required features F^+
- *a set of* prohibited features F^-
- *a set of* roles $R = [R_1, \ldots, R_k]$
- *a* role-constraint *formula* $\phi[R_1, \ldots, R_k]$

For example, the following activity specification describes the class of plan instances with the feature *Survey* but not *Heuristic*, where the variables that fill the roles *Start* and *Destination* are instantiated to values in the same sector.

<{*Survey*}, {*Heuristic*}, {*Start, Destination*},
{*(= (sector Start) (sector Destination))*}>

A *desire specification* constitutes the goal-oriented analogue of an activity specification, consisting of a collection of required features, prohibited features, roles, and role constraints for goals.

Agent Context Just as individual plans employ preconditions to limit their applicability, guidance requires a similar mechanism for defining scope. To this end, we introduce the notion of an *agent context*. While plan preconditions are generally limited to beliefs about the world state, our model of agent context focuses on the full operational state of an agent, characterized in terms of its beliefs, desires, and intentions. Beliefs are specified in terms of constraints on the current world state. Desires are specified as desire specifications describing goals that the agent has adopted. Intentions are specified as activity specifications describing plans currently in execution by the agent.

Our model of agency assumes a hierarchical collection of plans and goals; furthermore, agents are capable of multitasking (i.e., addressing multiple goals in parallel). Within a given phase of the BDI executor cycle, an agent's goals can be scoped in three ways:

- *Current goal:* the goal for which the BDI interpreter is selecting a plan to execute
- *Local goals:* the current goal or any of its ancestors
- *Global goals:* any goal of the agent

By distinguishing these different scopes for goals, guidance can be localized to more specific situations. Plans being executed can be scoped in a similar fashion.

Definition 2 (Agent Context) *An* agent context *is defined by a tuple* $\kappa = \langle \Phi, \Delta, A \rangle$*, where*

- Φ *is a set of well-formed formulae.*
- $\Delta = \Delta^C \cup \Delta^L \cup \Delta^G$ *is a set of current, local, and global desire specifications, respectively.*
- $A = A^L \cup A^G$ *is a set of local and global activity specifications, respectively.*[1]

Strategy Preference Strategy preference guidance consists of two components: an *agent context* and a *response activity specification*. The activity specification designates the class of recommended plan instances to be applied (i.e., choice of plan and variable instantiations for designated roles) when the agent enters a state that matches the designated agent context.

Definition 3 (Strategy Preference) *A* strategy preference *rule is defined by a pair* $\langle \kappa, \alpha \rangle$ *where* κ *is an agent context and* α *is an activity specification.*

Example

To illustrate strategy preference guidance, we consider an example from a simplified description of the TIGER domain.

Within TIGER, the demand for intelligence gathering and other services generally exceeds the capabilities of the available agents. As a result, task management constitutes one of the key functions of TIGER agents. We assume that tasks have associated properties such as type (e.g., survey, evacuation, medical emergency) and priority.

A TIGER agent controls a single vehicle (either a truck or a helicopter) and can work on at most one task at any time. When an agent receives a task in the form of a goal (*task t*), it must decide whether to start on the task immediately, to postpone the task until other tasks are completed, or to drop the task, leaving it for other agents to perform. We use the following predicates to represent the task execution state of the agent:

- (*available*) – the vehicle that the agent is controlling is available for use
- (*doing t*) – the agent is doing task *t*
- (*pending t*) – the agent will do task *t* after it has completed other tasks
- (*ignored t*) – the agent has determined not to do task *t*

Using a simple notation for describing plans, Figure 1 defines three plans for responding to the goal (*task t*). In the bodies of these plans, we represent the action of asserting a fact ϕ into the agent's belief set by $+\phi$ and retracting ϕ by $-\phi$. We use the notation $\{a_1, \ldots a_k\}$ to represent performing k actions in parallel within the body of a plan. The notation $a_1; a_2$ denotes action sequencing.

The first plan encodes a response to (*task t*) when not engaged in another task; it involves asserting (*doing t*), retracting (*available*), and then performing *t*. The second plan records the task as pending in the case where the agent is busy. The third plan ignores the task. Additional plans (not shown) provide the capabilities to start a pending task when the active task completes, to perform a task, etc.

Features for these plans reflect their inherent semantic differences: *Adopt* indicates that the plan results in the agent

[1]Because the motivation for guidance is to influence the choice of plan for the current goal, the agent context excludes an activity specification for the current plan.

Name: *Immediate-Response*
Cue: $(task\ t)$
Preconditions: $(available)$
Body: $\{-(available), +(doing\ t)\}; (do\ t)$
Features: *Adopt* Roles: $NewTask = t$

Name: *Delay-Response*
Cue: $(task\ t)$
Preconditions: $\neg(available)$
Body: $+(pending\ t)$
Features: *Adopt*,*Delay* Roles: $NewTask = t$

Name: *Ignore-Response*
Cue: $(task\ t)$
Preconditions: *true*
Body: $+(ignored\ t)$
Features: *Ignore* Roles: $NewTask = t$

Figure 1: TIGER Task Management Plans

deciding to perform the goal task, although maybe not immediately; *Delay* indicates that the plan results in execution of *t* being delayed; *Ignore* indicates that the plan results in the agent deciding not to perform *t* (the complement of *Adopt*). The role *NewTask* is used to reference the new task under consideration. For goal $(task\ t)$ we associate the feature *TaskResponse* and role $CurrentTask = t$.

With these features and roles, we can define strategy preference rules to provide guidance for responding to a new task. For example, the guidance "Adopt medical emergency tasks that involve more than 5 people" could be represented by the following strategy preference rule:

```
Agent Context:
    Current Desire Specification:
        Features+: TaskResponse
        Roles: CurrentTask
        Constraint:
            (and (= (task-type CurrentTask) medical)
                 (> (task-number-affected CurrentTask) 5))
Response Activity Specification:
    Features+: Adopt
```

The agent context states that the guidance is applicable when the current goal has the feature *TaskResponse* and the role *CurrentTask*, so that the task that instantiates *CurrentTask* has type *medical* and more than five people affected. The response activity specification designates any plan with the feature *Adopt*. Given the plans defined above, that would mean either *Immediate-Response* or *Delay-Response*.

Guidance Semantic Model

Space limitations preclude a full description of the semantics for guidance satisfaction defined in (Myers & Morley 2002). We present a brief summary here.

The semantic model for guidance satisfaction interprets strategy preference rules as a filter on the plan instances that an agent can execute. A strategy preference rule is deemed to be *relevant* to a given BDI executor cycle iff the agent context of the rule matches the current execution state. For a relevant rule to be satisfied, the BDI interpreter must not select a plan instance that violates the rule's activity specification. One interesting consequence of this model is that a strategy preference rule that is not *relevant* to the current decision cycle is trivially satisfied.

When a BDI agent needs to expand a goal, it determines the *applicable plans* for the goal and selects one of these to add to its intentions. Under the guidance satisfaction model, the set of plans from which the agent selects is restricted to those that satisfy the strategy preference rules: the agent identifies the relevant strategy preference rules and filters out plan instances that do not match the suggested response. The BDI interpreter selects one of the remaining plans to execute for the current goal.

Types of Guidance Conflict

Guidance can lead to two types of conflict: *plan selection* and *situated guidance*.

Plan Selection Conflict

A plan selection conflict occurs when multiple pieces of guidance make incompatible recommendations for responding to a goal within a given cycle of the BDI executor. Conflicts of this type can arise in different forms. Here, we distinguish between *direct* and *indirect* conflicts.

A *direct conflict* arises when guidance yields contradictory plan selection recommendations. At the plan level, such conflicts can arise through explicitly contradictory directives (e.g., guidance that reduces to the constraints *Execute plan P* and *Don't execute plan P*), or implicitly because of in-place control policies (e.g., guidance that reduces to the constraints *Execute plan P* and *Execute plan Q* in the context of a control policy that allows only one response to any posted goal). Conflicts can also arise at the level of variable bindings (e.g., *Instantiate role R to A* and *Instantiate role R to B*, where $A \neq B$).

An *indirect conflict* among guidance occurs when there is no direct conflict, yet the plans recommended by the guidance cannot complete successfully because of interplan interactions. Such a situation could arise due to future contention for resources, deadlock/livelock, or race conditions (among others). The problem of indirect conflict arises for any multithreaded system, not just systems in which guidance has been used to select activities.

Direct conflicts are easy to detect, as they lead to incompatible recommendations for responding to a posted goal. In contrast, it is generally difficult to detect *a priori* the plan interference problems that underlie indirect conflicts. Because such interference problems remain an open research area in the agents community, we focus exclusively on direct conflicts in the remainder of this paper.

Situated Guidance Conflicts

The semantic model from (Myers & Morley 2002) interprets guidance as a filter on the set of otherwise applicable plan instances for a particular goal or event. For example, consider a situation in which all TIGER vehicles are in use for various tasks, and the human supervisor has asserted the guidance Adopt medical emergency tasks that involve more than 5 people from the previous section. Suppose an emergency event arises. The relevant task adoption plans (namely *Immediate-Response* and *Delay-Response*) each require the availability of a vehicle. Because all vehicles are in use, no immediate response plans could be adopted for the event. Had there been a vehicle available, however, an emergency response of some form would have been adopted. Furthermore, according to the filtering semantic model, the declared guidance will eliminate from consideration the only applicable response, namely *Ignore-Response*, because it does not satisfy the guidance recommendations.

In this case, there is a clear expectation on the part of the human supervisor for the system to adopt a task in response to the emergency. Supporting this reaction requires a generalization of the filter-based semantic model described above.

More generally, this type of conflict arises in situations where a plan p is relevant for a current goal g but some precondition C of p does not hold, making p inapplicable. The unsatisfied condition may be blocked by a contradictory belief of the agent, or some already executing activity. This type of situation can arise independent of guidance. Our interest in such situations relates to cases where guidance would recommend the execution of p but the violation of C eliminates its consideration. In other words, the prior activity or state condition conflicts with the intent of applying the guidance. For this reason, we call this phenomenon a *situated guidance conflict*, as it depends on the consideration of guidance within a particular execution state of an agent.

In certain situations, there may be no recourse to address the violated conditions (e.g., consider a requirement for favorable weather). However, others may be *resolved* by undertaking appropriate actions in the domain. Proactive response of this type lies at the heart of our techniques for resolving for this class of guidance conflict.

Conflict Resolution

The original semantic model for guidance interpretation has a *passive* flavor in that it simply filters otherwise applicable plan instances that violate current guidance. This passive semantic model has the virtue of simplicity, but it eliminates the applicability of guidance in many interesting situations.

One problematic situation arises when guidance makes contradictory recommendations (e.g., *"Execute plan P"* and *"Don't execute plan P"*) as described above for the case of plan selection conflicts. Because the passive semantic model eliminates plans that violate current guidance, such a situation would lead to the selection of no plan. As noted above, the filter-based semantics also leads to trivial satisfaction of guidance in cases where a more proactive interpretation would be preferred.

Meaningful resolution of guidance conflicts requires a richer semantic model for guidance satisfaction. Our approach builds on the definition of satisfaction of an individual piece of guidance from (Myers & Morley 2002). However, we adopt a preference-based approach that seeks to maximize guidance satisfaction relative to stated priorities. Furthermore, instead of reducing the set of plans that the agent considers (by filtering plans that violate guidance), we expand the set to include options that would otherwise be discarded as inapplicable in the current execution state.

Preference Semantics for Plan Selection

Our approach to resolving plan selection conflicts involves identifying a plan instance that 'best satisfies' current guidance, through the definition of a partial order over plan instances. We assume that strategy preference rules include a *weight* reflecting the relative strength for that preference.

Different criteria could be used for combining and comparing the weights associated with the strategy preference rules to produce the partial order. We adopt an approach that rewards guidance satisfaction while punishing guidance violation, as characterized by the following *guidance ranking function* for plan instances.

Definition 4 (Guidance Ranking of a Plan) *Let p be a plan instance for a goal g and Q be the current set of guidance, where $Q_p^+ \subseteq Q$ is the set of guidance satisfied by p and $Q_p^- \subseteq Q$ is the set violated by p. The* guidance ranking *of p is defined as follows in terms of the priority $GPriority(q)$ associated with each guidance rule q:*

$$GRanking(p) = \sum_{q \in Q_p^+} GPriority(q) - \sum_{q \in Q_p^-} GPriority(q)$$

Candidate Plan Expansion

The ranking of applicable plan instances is insufficient to resolve situated guidance conflicts. Our approach involves expanding the set of plan instances considered for application to a given goal. This expanded set builds on the agent's library of predefined plans, extending plans that guidance might recommend to compensate for violated applicability conditions. The expansion process requires the satisfaction of certain prerequisites related to *resolvability* and *cost-benefit analysis*.

Resolvability The unsatisfied applicability conditions of the guidance-recommended plan must be *resolvable*. In particular, there must be identified methods (called *resolution plans*) that can be invoked to achieve the unsatisfied conditions.

Cost-Benefit Analysis The benefits in following the prescribed guidance must outweigh the costs associated with executing the resolution plans.

We consider these two requirements in turn, and then describe the process for expanding the set of plan instances to be considered for a given goal.

Resolvability Resolution plans could be defined for a wide range of conditions. Resource availability constitutes one important class; in this paper, we focus on conditions related to serially sharable resources (i.e., resources that can be used sequentially but not simultaneously).

In this context, resolution plans must free the resource employed by the current activity to enable its use by the guidance-recommended plan. Within the context of serially sharable resources, *cancellation* of the prior activity constitutes one obvious solution. In addition, prior activities could be modified to eliminate the dependence on the conflicted resource, say by *substituting* a different resource or by *delaying* the prior activity. We call a current activity that is impacted by a resolution plan a *conflict task*.

In addition to establishing resolvability conditions, resolution plans must also leave an agent in a coherent state. This requirement means that a resolution plan must consider the in-progress effects of any current activities that are to be modified or canceled to ensure that appropriate 'clean-up' processes are invoked. For example, consider an emergency response task in which a truck has picked up a set of supplies to deliver to a designated location. Early termination of such an activity might involve, at a minimum, delivering those supplies to a local depot (for delivery by some other agent). Definition of *recovery mechanisms* of this type is not restricted to agents under human supervision via guidance, but rather constitutes a problem for any agents that need to dynamically modify their activities at runtime.

Checkpoint schemes constitute one standard technique for ensuring state coherence when activities may need to be terminated prematurely. With checkpoint schemes, coherent states are saved periodically to enable rollback to a consistent state in case of unrecoverable failures. Because agents operating in dynamic environments will generally perform activities that change the world in irrevocable ways, such schemes are not viable. Instead, agents require *forward* recovery methods that can take actions to transition from an unable situation to some known, safe state.

Formulation of forward recovery methods presents several problems. In general, the specific actions to take could depend both on the state of execution of the in-progress plans and on the status of other executing activities and world state properties. In the worst case, a unique recovery method would be required for each such situation. For this reason, forward recovery mechanisms for agents are generally implemented in an *ad hoc*, domain-specific manner.

The resolution plans for task management within TIGER were defined by hand. They consist of plans for *delaying* and *terminating* prior tasks. Because the number of subtasks involved with the survey and emergency response tasks is relatively small (i.e., fewer than five), the number of possible combinations of state to consider is correspondingly low. Furthermore, tasks are undertaken independently of each other, so cross-task interactions were not an issue.

Cost-Benefit Analysis The value in modifying existing activities to enable activation of new guidance-recommended activities depends on a range of factors. We consider a model grounded in the following two concepts:

- *Resolvability Cost:* the cost of executing any required resolution plans
- *Activity Priorities:* the priorities of the recommended activity and any *conflict activities*

Different approaches can be considered for combining the above factors to determine the appropriate response to a situated guidance conflict. We describe three general approaches here.

One approach involves defining a multidimensional objective function that determines when the guidance-recommended activity should be undertaken. Such a function would cover every possible situated guidance conflict, thus enabling a fully automated conflict resolution method. Multidimensional evaluation functions are notoriously difficult to define, as they must relate values that are not directly comparable. The need to consider all combinations of values complicates matters further.

In contrast, a mixed-initiative approach could be adopted in which a human supervisor determines the appropriate course of action based on the factors cited above. This approach avoids the cost of defining *a priori* comparison functions and provides maximum flexibility at runtime. However, situational conflicts may arise frequently in some domains, thus burdening the user with a high level of decision making involvement.

A third approach involves the definition of policies (similar to guidance) for determining the conditions under which modification of earlier activities should be undertaken. Such policies would involve constraints on the various factors described above. For example, "Only modify ongoing activities when the priority of the new task exceeds that of the original, and the recovery cost is less than 0.5". For situations where the policies are sufficient to identify a response, conflicts will be resolved automatically. In other cases, the human supervisor would be engaged to make the appropriate decision. As such, this third approach combines the benefits of predefining certain responses with the flexibility of runtime decision-making by the human supervisor for situations that cannot be readily characterized ahead of time.

Expanded Set of Candidate Plans Our approach to expanding the set of candidate plans for a goal *g* involves the dynamic creation of a set of *proactive plans*. Each proactive plan is a variant of a *potentially applicable plan* – a relevant plan for *g* whose applicability is blocked by one or more unsatisfied but resolvable preconditions. The body of the proactive plan incorporates activities from appropriate resolution plans to achieve the blocked preconditions, as well as the actions from the potentially applicable plan. In this way, proactive plans extend the set of actions that can be taken by an agent for a given goal. The following definitions capture these notions more precisely.

Definition 5 (Resolvable Condition) *A condition* ϕ *is* resolvable *in a given BDI executor state iff there is some instance* p^r *of a resolution plan such that* $Cue(p^r) = \phi$ *and for every* $\phi' \in Pre(p^r)$, *Believed*(ϕ') *holds in the BDI state.*

Definition 6 (Potentially Applicable Plan Instance) *A* potentially applicable plan instance *for a goal g is a relevant plan instance for g for which not all preconditions are satisfied but all unsatisfied preconditions are resolvable.*

Definition 7 (Proactive Plan) *Let p be a potentially applicable plan instance with* $Cue(p) = g$. *Let* $Pre(p) = \Phi^F \cup \Phi^T$ *where* $\Phi^F = \{\phi_1^F, \ldots, \phi_m^F\}$ *are unsatisfied and* $\Phi^T = \{\phi_1^T \ldots \phi_n^T\}$ *are satisfied. Let* $p_1^r, \ldots p_m^r$ *be resolution plans such*

Name: *Resolve-by-Delay*
Cue: $(available)$
Preconditions: $(doing\ t')$
Body:
$(pause\ t'); \{+(available), -(doing\ t'), +(pending\ t')\}$
Features: *DelayCurrent* Roles: $CurrentTask = t'$

Name: *Resolve-by-Termination*
Cue: $(available)$
Preconditions: $(doing\ t')$
Body: $(kill\ t'); \{+(available), -(doing\ t'), +(ignored\ t')\}$
Features: *DropCurrent* Roles: $CurrentTask = t'$

Figure 2: TIGER Resolution Plans

that $Cue(p^r_i) = \phi^F_i$ *and for every* $\phi \in Pre(p^r_i)$, $Believed(\phi)$ *holds in the current BDI state. The plan* p' *defined below is a proactive plan for g.*

- $Cue(p') = g$
- $Pre(p') = \Phi^T \cup \bigcup_{1 \le i \le m} Pre(p^r_i)$
- $Body(p') = \{Body(p^r_1), \ldots, Body(p^r_m)\}; Body(p)$
- $Features(p') = Features(p) \cup \bigcup_{1 \le i \le m} Features(p^r_i)$
- $Roles(p') = Roles(p) \cup \bigcup_{1 \le i \le m} Roles(p^r_i)$.

Definition 8 (Proactive Plan Set) *The* proactive plan set *for goal g, denoted by Proactive*(g)*, consists of the proactive plans that can be constructed from the potentially applicable plan instances for g and the available resolution plans.*

Note that the applicability of a proactive plan within the current BDI executor loop is guaranteed, as its preconditions are drawn from a set of known satisfied conditions.

As noted above, cost-benefit considerations should be taken into account when deciding how to augment the original set of applicable plan instances for a current goal with proactive plans. Thus, in general, only a subset of the set *Proactive*(g) of proactive plans for g would be included.

Given the expanded set of candidate plans for application to the current goal g, selection among them can be performed in accord with the preference semantics outlined earlier, through application of the guidance ranking function of Definition 4.

TIGER Conflict Resolution

TIGER supports situated conflict resolution for task management activities. Within this context, resource contention arises because each vehicle is limited to use on at most one task at a time. TIGER includes resolution plans (see Figure 2) that achieve $(available)$ (i.e., the availability of the vehicle) by delaying and terminating prior tasks, thereby enabling a new task to be executed immediately. The resolution plan *Resolve-by-Delay* delays the current task t' via the goal $(pause\ t')$. The resolution plan *Resolve-by-Termination* terminates the current task via the goal $(kill\ t')$. The goals $(pause\ t')$ and $(kill\ t')$ also perform appropriate clean-up actions. When an agent is given a new task but $(available)$ is unsatisfied, it can synthesize two proactive plans (shown in Figure 3) from the plan *Immediate-Response* and the resolution plans of Figure 2.

Name: *Proactive-Immediate-Response-Delay*
Cue: $(task\ t)$
Preconditions: $(doing\ t')$
Body: $(pause\ t'); \{+(available), -(doing\ t'), +(pending\ t')\};$
$\{-(available), +(doing\ t)\}; (do\ t)$
Features: *Adopt*, *DelayCurrent* Roles: $NewTask = t, CurrentTask = t'$

Name: *Proactive-Immediate-Response-Termination*
Cue: $(task\ t)$
Preconditions: $(doing\ t')$
Body: $(kill\ t'); \{+(available), -(doing\ t'), +(ignored\ t')\};$
$\{-(available), +(doing\ t)\}$
Features: *Adopt*, *DropCurrent* Roles: $NewTask = t, CurrentTask = t'$

Figure 3: Sample Proactive Plans

For the small number of plans in this example, resolution rules might seem unnecessary – one can achieve the same effect by extending the agent's plan library to include the proactive plans. However, with a greater number of plans, the ability to factor out the recovery mechanisms eliminates duplication and reduces the potential for error. Alternatively, one could add explicit subgoals of achieving $(available)$ (in this case) into the base plans. Because the choice of method for achieving $(available)$ would occur after the choice of the base plan, this approach would preclude the use of guidance to select among options.

The cost-benefit analysis used by TIGER for determining which proactive plans to consider for a given goal consists of the following three conditions. First, resolvability costs are computed as a simple heuristic related to expected time remaining for task completion; a threshold is defined so that nearly complete tasks are not interrupted. Second, the priority of the new task must be at least as high as that of the resolvable task. Third, there must be at least one current strategy preference rule whose agent context is satisfied and whose activity specification matches the proactive plan. In the future, we intend to replace this fixed criteria with a policy-based approach similar to that described earlier.

Related Work

Most work on conflict within the agents community has focused on conflicts *among* agents (e.g., the papers in (Tessier, Chaudron, & Muller 2000)) rather than on guidance for controlling an agent. Typically, explicit interagent protocols are used to negotiate solutions to detected conflicts.

Conflicting advice has been considered previously in the context of advising an automated planning system (Myers 2000b). Detection and resolution techniques in that work have a markedly different style from the approach outlined in this paper, being grounded in heuristic search control methods for plan generation.

The rule-based reasoning community has considered a similar problem dealing with resolving rule conflicts. In

(Chomicki, Lobo, & Naqvi 2000), rule conflicts are resolved by ignoring triggering events that cause conflicts. The work of (Jagadish, Mendelzon, & Mumick 1996) uses a declarative set of metarules to constrain how a set of rules should be executed. Their metacontrol language is grounded in the specifics of rules and rule firing (e.g., ordering rules, disabling rules, requiring simultaneous triggering of rules), which differs substantially from our more expressive and user-focused guidance language.

The work in (Lupu & Sloman 1999) presents a simple language for defining policies to manage a distributed set of objects. Static conflict checking is performed when policies are defined (rather than at runtime, as in this paper). Their conflict resolution methods include straightforward concepts such as explicit priorities and preferring negative to positive rules; in addition, they incorporate more advanced notions of specificity over rules and distance metrics to generate overall precedence relationships.

In (Dignum *et al.* 2000), preferences over BDI agent behaviors are expressed through a multimodal deontic logic for social norms and obligations. Metalevel norms and obligations are used to resolve conflicts that arise.

The dialog community has focused on techniques for detecting and resolving conflicts that arise in the performance of collaborative tasks (e.g., (Qu & Beale 1999)). Techniques of this type could be used as the basis for interactive resolution of the conflict types explored in this paper.

Conclusions

Many applications that could benefit from agent technology impose the requirement that humans retain control over agent operations. The guidance framework of (Myers & Morley 2001; 2002) enables a human supervisor to assert strategy preference rules to influence agent behavior, thus providing a powerful mechanism for directing agent operations. However, it also introduces the possibility for problems in the event that conflicting guidance is declared.

This paper identified two classes of guidance conflict, namely *plan selection* and *situated guidance*, and showed how such conflicts transcend the purely filter-based semantics introduced originally for agent guidance. To address such conflicts, we defined a generalized semantic model for guidance satisfaction that incorporates the complementary notions of guidance-derived plan preference relations and plan option expansion. Within this model, the set of plan options available to an agent is extended to include plans that would not normally be considered for execution. These options can be synthesized dynamically by combining guidance-relevant plans whose applicability is blocked with resolution plans that can achieve the blocked applicability conditions. Accompanying conflict resolution methods for this model were presented that ensure robustness of agent operations in the face of conflicting guidance.

Acknowledgments This work was supported by DARPA under the supervision of Air Force Research Laboratory contract F30602-98-C-0160.

References

Chomicki, J.; Lobo, J.; and Naqvi, S. 2000. A logic programming approach to conflict resolution in policy management. In Cohn, A. G.; Giunchiglia, F.; and Selman, B., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '00).* Morgan Kaufmann Publishers.

Dignum, F.; Morley, D.; Sonenberg, E. A.; and Cavedon, L. 2000. Towards socially sophisticated BDI agents. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS'2000).*

Georgeff, M. P., and Ingrand, F. F. 1989. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*.

Jagadish, H. V.; Mendelzon, A. O.; and Mumick, I. S. 1996. Managing conflicts between rules. In *Proceedings of the ACM Symposium on Principles of Database Systems*.

Lupu, E., and Sloman, M. 1999. Conflicts in policy-based distributed systems. *IEEE Transactions on Software Engineering, Special Issue on Inconsistency Management* 25(6).

Myers, K. L., and Morley, D. N. 2001. Human directability of agents. In *Proceedings of the First International Conference on Knowledge Capture*.

Myers, K. L., and Morley, D. N. 2002. Policy-based agent directability. In Hexmoor, H.; Falcone, R.; and Castelfranchi, C., eds., *Agent Autonomy*. Kluwer Academic Publishers.

Myers, K. L. 1996. Strategic advice for hierarchical planners. In Aiello, L. C.; Doyle, J.; and Shapiro, S. C., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96).* Morgan Kaufmann Publishers.

Myers, K. L. 2000a. Domain metatheories: Enabling user-centric planning. In *Proceedings of the AAAI-2000 Workshop on Representational Issues for Real-World Planning Systems*.

Myers, K. L. 2000b. Planning with conflicting advice. In *Proceedings of the Fifth International Conference on AI Planning Systems*.

Qu, Y., and Beale, S. 1999. A constraint-based model for cooperative response generation in information dialogues. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*.

Rao, A. S., and Georgeff, M. P. 1995. BDI agents: From theory to practice. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*.

Tessier, C.; Chaudron, L.; and Muller, H.-J., eds. 2000. *Conflicting Agents*. Kluwer Academic Press.

Wilkins, D. E., and Myers, K. L. 1995. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation* 5(6).